SEISMIC PROCESSING VIA MACHINE LEARNING FOR EVENT DETECTION AND PHASE PICKING

A Dissertation
Presented to
The Academic Faculty

By

Lijun Zhu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2019

SEISMIC PROCESSING VIA MACHINE LEARNING FOR EVENT DETECTION AND PHASE PICKING

Approved by:

Dr. James H. McClellan, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ghassan AlRegib
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Zhigang Peng
School of Earth and Atmosphere Science
Georgia Institute of Technology

Dr. David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Andrew V. Newman
School of Earth and Atmosphere Science
Georgia Institute of Technology

Date Approved: July 9, 2019

[Computers] but they are useless. They can only give you answers.

Pablo Picasso

This dissertation is dedicated to

May, our son, and my parents.

ACKNOWLEDGEMENTS

Looking back at the ten-year journey I had at Georgia Tech, none of the progress and achievements I made will ever be possible without the support, guidance, and inspiration from professors, peer graduate students, mentors during my internships, as well as my families.

I would like to first thank my Ph.D. advisor, Prof. James McClellan for the continuous support of my study and research. Ever since the first lecture of the undergraduate signal processing class in my freshman year, Prof. McClellan has been an excellent mentor of both my academic and career paths. Inspired by his enlightening teaching and the encouraging guidance in the vertically integrated programs (VIP), decided to dedicate my career to the study of signal processing. As my advisor, Prof. McClellan encouraged me to explore a wide range of possibilities and generously supported both my successes and fails. Without his patience and understanding, I would never have gone this far in this thesis. I want to present my sincere gratitude to Prof. Ghassan AlRegib for his long-term supporting as a member of the Center for Energy and Geo Processing (CeGP). I also would like to thank Prof. Zhigang Peng for his guidance in the seismic processing as well as the ample support on high-quality seismic waveform data. I appreciate Prof. David Anderson and Prof. Andrew Newman, who spend their precious times in serving in my dissertation committee and provide me valuable feedback on my thesis works.

Many thanks to the postdoctoral fellow, peer graduate students, and my mentors from previous internships for the inspiring discussions, collaboration, and guidance. Dr. Entao Liu, a former postdoctoral fellow in CeGP, guided me in the first four years of my Ph.D. program, who kindly gave me advice and extraordinary insights in algorithm design and paper writing. I also appreciate Dr. Zefeng Li for his abundant knowledge in processing seismic waveform data and our collaboration on the local similarity scheme. I would like to thank Dr. Dongdong Yao for teaching me details on the matched filter technique and

processing the Wenchuan dataset; Chenyu Li for conducting transfer learning experiments for CPIC models; Dr. Yangfan Deng and Dr. Qiu Zhong for testing CPIC model on the Mariana dataset; and Dr. Zhiling Long, Dr. Zhen Wan, Dr. Yuting Hu, Dr. Yazeed Alaudah, Motarz Alfarraj, Liangbei Xu, Lindsay Chuang, Qiushi Zhai for helpful discussions. I would also like to thank the mentors in my internship, Guy Torio. Jahn Eichfeld, Dinei Florencio, Dr. Weichang Li, Dr. Yang Zhao, Dr. Arthur Redfern, and Dr. Tarek Lahlou, who helped me obtain the first-hand experience in applying research results on the enterprise applications.

Last but not least, I would like to thank my families for unconditional love and support for my life and career. Without their understanding, encouragement, and sacrifice, the completion of my works would not have been possible. Many thanks to my friends and who assisted me in the past years for making this long journey the most memorable experience in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

Adam	adaptive moment estimation.
ADC	analog-to-digital converter.
AIC	Akaike information criterion.
AR	autoregressive.
AR-AIC	phase picking using AIC on AR models.
ARM	Advanced RISC Machine.
BLAS	basic linear algebra subprograms.
BN layer	batch normalization layer.
CF	characteristic function.
CNN	convolutional neural network.
CONV layer	convolutional layer.
CPIC	CNN-based phase identification classifier.
EEW	earthquake early warning.
EF	envelope function.
FC layer	fully-connected layer.
FFT	fast Fourier transformation.
FWL	Finite Word Length.
GAN	generative adversarial network.
GEMM	general matrix multiply.
GPU	graphics processing unit.
GSM	global system for mobile communications.
ILSVRC	ImageNet large scale visual recognition challenge.
IoT	internet of things.
LMS	least mean square.
LSTM	long short-term memory.
M_L	local magnitude.

M_W	moment magnitude.
MAC	multiply and accumulate.
MAD	median absolute deviation.
MEMS	microelectromechanical systems.
MLE	maximum likelihood estimation.
MLP	multilayer perceptron.
NCS	neural compute stick.
NN	neural network.
ObsPy	an open-source Python framework for processing seismological data.
P-wave	primary wave or pressure wave.
Python	an interpreted, high-level, general-purpose programming language.
PyTorch	an open-source deep learning platform.
QC	quality control.
RAM	random-access memory.
ReLU	rectified linear unit.
RNN	recurrent neural network.
S-wave	secondary wave or shear wave.
SCSN	Southern California Seismic Network.
SGD	stochastic gradient descent.
SIMD	single instruction multiple data.
SNR	signal-to-noise ratio.
SoC	system on chip.
STA/LTA	short-term average over long-term average ratio.
SVM	support vector machine.
TDNN	time-delay neural networks.
USB	universal serial bus.
USGS	United States geological survey.
UTC	coordinated universal time.

SUMMARY

A feasible solution for seismic event detection and phase picking is prototyped on an embedded system with seismic sensors using a lightweight convolutional neural network (CNN). Event detection and phase picking are essential for locating seismic events and imaging subsurface structures. Real-time detection and picking using an embedded system with seismic sensors are not only valuable for time-sensitive tasks, e.g., earthquake early warning (EEW) systems but also provide the foundation of an interconnected smart sensor network, e.g., internet of things (IoT), for modern seismic acquisition and monitoring. However, existing detection and picking methods are either too simple to achieve the desired accuracy or too computationally complex to be deployed on an embedded system along with the seismic sensors. Accurate offline training of the CNN is demonstrated for small and large datasets, while observing the trade-off between accuracy and computation cost, which gives an efficient online deployment of the neural network. When trained for a small dataset from one area, transfer learning is used to modify the CNN parameters with modest retraining in order to generalize and validate the CNN model for processing in other regions. Simplification of the model and quantization of its parameters is explored to develop a prototype that is suitable for embedded devices. The product of this research is a universal seismic event detection and phase picking tool that is accurate and efficient for processing large volumes of data, as well as lightweight for deployment on an embedded system.

CHAPTER 1
INTRODUCTION

When an earthquake waveform, or a seismogram, is recorded and analyzed, much can be inferred simply from its arrival times, when the elastic waves first arrived on multiple recording devices. The relative difference between the P-wave and S-wave arrival times on a single station depends on the distance the elastic wave traveled between the source and receiver. A collection of relative arrival times on an array of receivers can be used to determine the source location via a travel-time inverse problem. A cloud of those event location points is then used to infer the attributes of faults or other geological structures. All these usages of arrival times are made possible if an earthquake event is correctly detected and its arrival time is estimated accurately. Historically, this was primarily done via manually picking the arrival times on each seismic trace by experienced seismologists. After computers were introduced into processing seismic waveform data in the late 1960s, automatic phase picking algorithms were proposed to assist in the manual picking, such as Allen [1], Morita and Hamaguchi [2], and Sleeman and Eck [3]. Recent studies [4, 5, 6] show that neural network (NN) based approaches can significantly improve the accuracy of the automatic detection and picking of arrival times given sufficient training samples. However, large training sets are not always available, especially in small or new monitoring regions, so Zhu *et al.* [7] explored the possibility of using NNs on small training sets. Furthermore, this work demonstrated that a well-trained NN model on a large dataset in one region can be transferred to another region with some tweaking of parameters on a small additional sample set [8]. These advances make the deployment of NNs on an embedded system for real-time field processing of seismic event detection and phase picking possible.

The development history of neural networks as well as seismic event detection and phase picking is reviewed in Section 1.1. Section 1.2 summarizes the shift of seismic

computing from the centralized cloud server to the distributed edge devices and proposes a new scheme combining the cloud and edge computing for deployment of neural network on seismic sensor networks. Last, the development of this new scheme is broken into two pieces in Section 1.3, which are covered in detail in Chapters 2 and 3.

1.1 A Peek into the History

1.1.1 Neural Networks

The history of the development of neural networks represents a typical top-down view of our understanding of the physical world: a theoretical hypothesis was made based on observations; systems based on the hypothesis were built to test it; difficulties and challenges forced improvements to the theory; and the methods used in testing inspired new theory and tools. Ironically, the very goal of machine learning is to offer an alternative approach to this top-down view by building a relationship between observations and the model directly without a specific model hypothesis, which is often regarded as a bottom-up view.

In 1943, based on the "all-or-none" character of neural activity, McCulloch and Pitts [9] proposed a propositional logic similar to electrical circuits to simulate the neuron's events and their relations, which was later referred to as "neural circuits." The model presented in McCulloch and Pitts [9] elegantly described the fundamental relationship and activities between neurons; however, it lacked a mechanism to properly learn the parameters in such neuron models. Inspired by the early work of Hebb [10], Rosenblatt [11] proposed the perceptron model in 1958. In 1949, Hebb [10] suggested a hugely influential idea that learning occurs primarily via the formation and breaking of synapses between neurons in the brain. The perceptron, which is the single-layer model shown in Figure 1.1, implemented the learning analogy by assigning weights to each input and passing the weighted sum into a gate function ("all-or-none") for a binary classification problem. The optimal model is achieved by adjusting those weights based on the difference between the current output and the expected output (from the training samples): increasing the weight if the

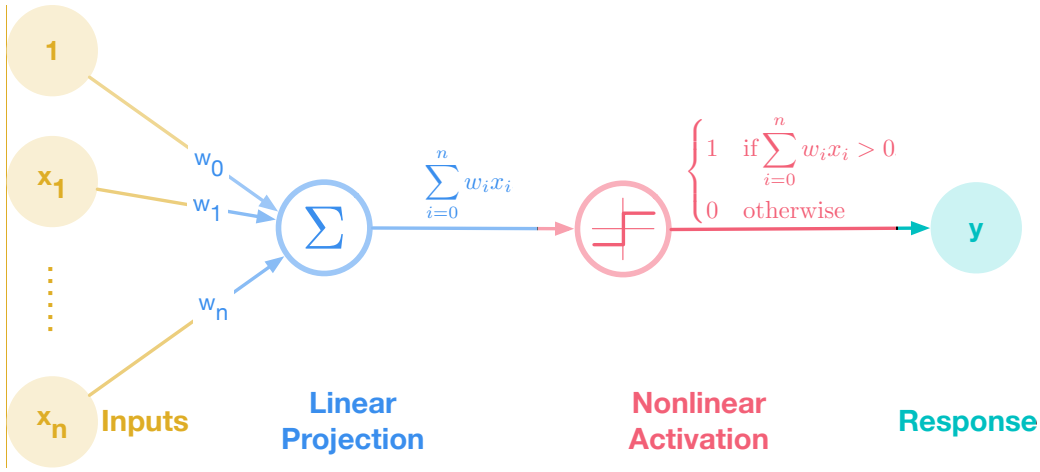


Figure 1.1: The organization structure of a perceptron with single output adapted from Rosenblatt [11].

current output is lower than the expected one, and vice versa. Simple as it may seem, such a structure, where a linear function is followed by a nonlinear activation function, is fundamental to all modern neural networks and is where they get the "neural" in their names. It is not surprising to envision a network of multiple perceptrons that can solve a multi-class classification problem, where they get the name of "network" of perceptrons, or neurons as they are usually referred to now. Unfortunately, the realistic technique of training multi-layer perceptron networks (later known as "backpropagation") was not proposed until the 1970s, and also rediscovered in the 1980s. Until then, most of the NN research focused on single-layer NN models. Although the perceptron has a biological analogy to the neural system, it is not the only way to form a single-layer neural network model.

In 1960, Widrow [12] suggested outputting the weighted input without a nonlinear activation function, which was named the "adaptive linear element" (Adaline). Similar to the adaptive filter, a simple least mean square (LMS) algorithm was used to solve an Adaline [13], which was later called the Widrow-Hoff delta rule [14]. Widrow and Hoff [15] showed that the learning mechanism is reduced to a minimization of output errors via derivatives. In fact, Adaline without the gate function is another form of linear regression. Widrow and Lehr [13] and his student devised Madaline Rule I (MRI) for many Adalines, the earliest

Table 1.1: Demonstration of linearly separable and inseparable functions.

(a) OR gate (separable)

Input A \ Input B	0	1
0	-	+
1	+	+

(b) XOR gate (inseparable)

Input A \ Input B	0	1
0	-	+
1	+	-

popular learning rule for NNs with multiple adaptive elements. Winter and Widrow [16] also proposed the improved Madaline Rule II (MRII) for a two-layer Madaline in 1988. The gate function was later replaced by a sigmoid function and the Madaline Rule III [17] was proposed in 1990, which was later found to be equivalent to the "backpropagation" algorithm [13].

The first documented hardware attempt to implement a NN model for a computing system dates back to 1956 [18], though Minsky [19] may have tried it years before. Unfortunately, both attempts were deemed unsuccessful. Although Von Neumann [20] himself suggested imitating neuron functions by telegraph relays or vacuum tubes, the traditional Von Neumann architecture [21] still took over the computing world. Moreover, Minsky and Papert [22] pointed out that single-layer perceptron is limited to linearly separable functions (Table 1.1a), e.g., an XOR gate cannot be properly represented (Table 1.1b). Minsky and Papert [22] further argued that only multiple layers of perceptrons are capable of representing more complex logic, which was later known as the multilayer perceptron (MLP). Unfortunately, it was widely agreed at the time that training the MLP model was not feasible due to the lack of an efficient training scheme as well as limited computing power. The same approach for training the single-layer perceptron is not generalizable to the multiple layers case: only the errors on the output layer can be properly computed. This lack of an error propagation mechanism was a problem first solved by Werbos [23] by propagating the error gradient using the chain rule (Figure 1.2), but his work was largely overlooked by the community. It was rediscovered independently by Parker [24] in 1982 and Rumelhart *et al.*

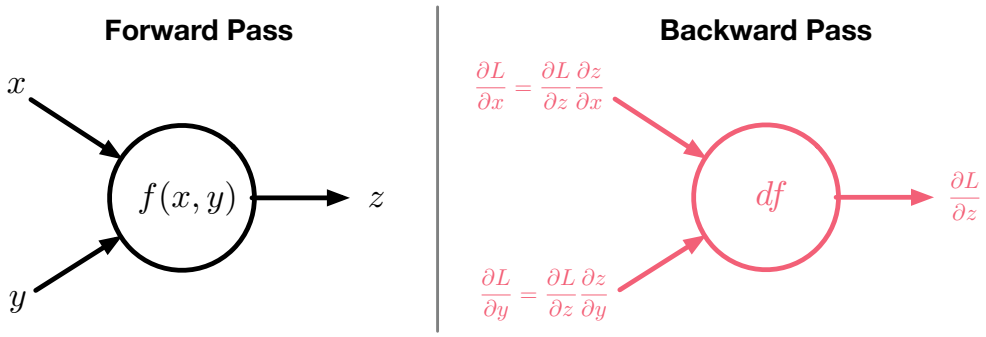


Figure 1.2: The chain rule for backpropagation: forward pass on the left with an arbitrary differentiable function $z = f(x, y)$, and backward pass on the right for the gradient of the loss function L with respect to the inputs x and y.

[14] in 1985. Rumelhart *et al.* [14] specifically address the problems raised by Minsky and Papert [22] and made it widely understood how MLPs could be trained for complex tasks. A mathematical proof that MLPs can theoretically implement any function, including an XOR gate, was later given by Hornik *et al.* [25].

The breakthrough of backpropagation inspired a series of advances in neural network applications for supervised and unsupervised learning. LeCun *et al.* [26] took the idea of subsampling, now known as a type of pooling, and weight sharing via convolutions [27] to develop a check-reading system based on convolutional neural network (CNN), and initiated the neural network applications for computer vision. Waibel *et al.* [28] proposed time-delay neural networks (TDNN) for speech recognition whose performance was later surpassed by recurrent neural network (RNN) via the discovery of long short-term memory (LSTM) [29]. Bourlard and Kamp [30] and Baldi and Hornik [31] explored the potential of NNs for unsupervised learning, and Hinton and Zemel [32] proposed the idea of an autoencoder to approximate the probabilistic distributions of the observable vectors. Other unsupervised learning trials include Kohonen [33] and Carpenter and Grossberg [34] on the topic of self-organizing maps. Ackley *et al.* [35] and Neal [36] worked on a neural network approach for learning probability distributions in the form of a Boltzmann machine and later results with the Helmholtz machine [37] improved the training efficiency [38]. Neural networks were also applied for reinforcement learning, such as the inverted pendulum

problem, balancing a stick on a moving platform [36], adaptive filtering [39], and robotics [40]. Although neural networks enjoyed success in backgammon [41], it proved to be too much of a challenge in the games of chess [42] and go [43] at that time.

The more demanding applications, once again, caught up to the computing limit of that time, which led to a widespread belief that deep neural networks were "no good and could never be trained" [44]. Research in neural networks halted for a long period in the late 1990s, but re-ignited after the famous publication of Hinton *et al.* [45] in 2006. More specifically, Hinton *et al.* [45] provided a clever way of initializing weights by training each layer one by one with unsupervised training, which is better than random values, and then finishing the supervised learning as neural networks proposed in Hinton [46]. In the next year, Bengio *et al.* [47] quickly followed up and presented a strong argument that deep networks are more efficient for difficult problems than shallow ones. It was recognized by [48] in the same year that having a deeper network, rather than better initialization, is the main reason Hinton *et al.* [45] succeeded. Nevertheless, Hinton *et al.* [45] started a new wave of NN research under the name of "deep learning", which inspired improvements in speech recognition [49] and later computer vision [50]. Algorithmic advances are certainly important, but another essential ingredient contributing to the success of deep learning is the pure computational power increase over the past decade. Coupled with the explosive growth of data recorded due to the "big data" movement, researchers in the 2010's have more tools at their disposal than their predecessors. This is amplified by a series of academic [51] as well as commercial [52] success in boosting the training of neural networks by graphics processing units (GPUs), and made the rectified linear unit (ReLU) (promoted by three groups [53, 54, 55]) the standard approach to mitigate the vanishing and exploding gradient problem [56]. From this point onward, deep learning required three standard ingredients: a large volume of training data, parallel computing via GPUs, and scalable models. Later developments to the theory were mostly on improving training stability and efficiency, such as Dropout [57] and batch normalization [58]. Now, we are in the era

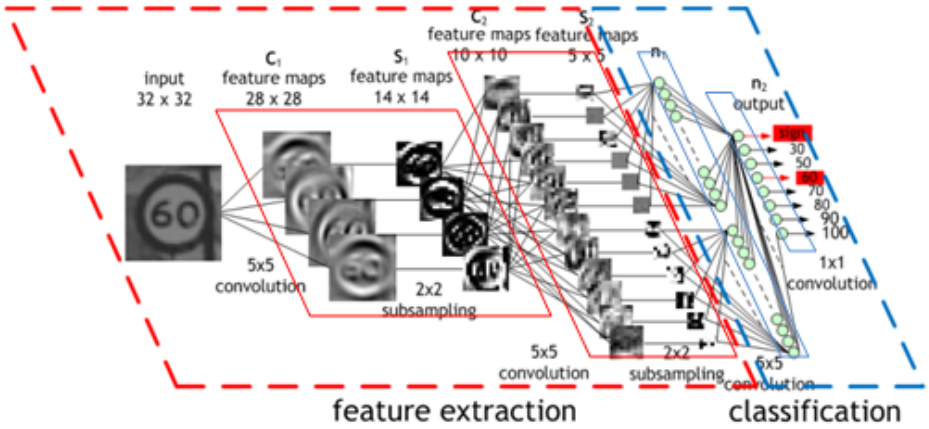


Figure 1.3: Visualization of a typical convolutional neural network structure for image classification tasks. Figure taken from `https://sites.google.com/site/5kk73gpu2013/assignment/cnn`.

of rapid and explosive growth in applications of the deep learning framework developed mostly before 2012.

Among different types of neural networks, the CNN has received tremendous attention and is rife in seismic applications [59] recently. A typical CNN structure is demonstrated in Figure 1.3. Originally developed for a computer vision task [26] in 1989, CNNs flourished in the pursuit of a better image classifier during the ImageNet large scale visual recognition challenge (ILSVRC) [60], in which a classifier is challenged to categorize 1,000 classes of images, given over one million training samples. In 2012, AlexNet [50] was the first well-known CNN model that was significantly better than the traditional approaches in classification accuracy. VGGNet [61] demonstrated that narrower but deeper model performance is better than the wide and shallow model in Krizhevsky *et al.* [50]. It was also one of the first models (along with GoogleNet [62]) that achieved less than 10% error rate on the ImageNet dataset in 2014. In the following year, ResNet [63] won the ImageNet challenge by providing shortcuts that make it possible to train a deeper network without degradation. ResNet also marked the historical point where a neural network model performed better than humans in the image classification task. Winners of the following ILSVRCs brought additional improved models such as Inception V4 [64] and SENet [65]; however, VGGNet

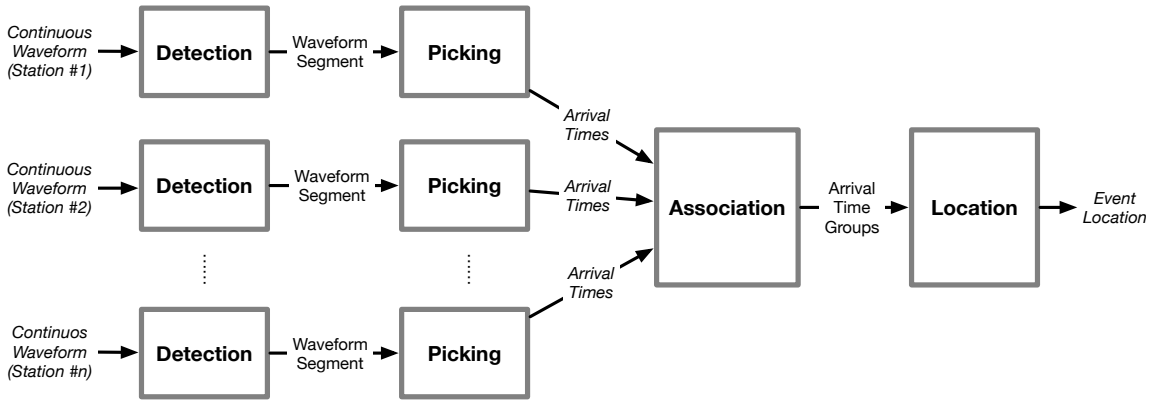


Figure 1.4: The processing pipeline of a typical seismic processing system. Detection and picking can be done on individual sensors, but association and location require communication among sensors.

and ResNet remain the standard models for most vision tasks in the industry.

1.1.2 Event Detection and Phase Picking

Modern seismic monitoring began in the late 19th century. The need for detecting a seismic event and picking arrival times did not show up until the 1960s when large volumes of continuous monitoring data became available, in part due to the nuclear explosion monitoring programs during the cold war. Shown in Figure 1.4, the recorded seismic waveforms are processed by experienced seismologists and the picked arrival times are used to infer additional properties of the seismic event, such as event origin location. With the explosive growth in the number of waveform recordings, seismologists must now rely on automatic event detection and phase picking algorithms. The fundamental task of event detection is to determine a time interval $[t_a, t_b]$ from a continuous time sequence $f(t)$, where the signal inside $[t_a, t_b]$ has some special properties of interest. Allen [1] introduced the short-term average over long-term average ratio (STA/LTA) in 1978 and later identified the two major components of every picking algorithm: a characteristic function (CF) and a peak detector [66]. As demonstrated in Figure 1.5, the event detection problem is formulated into detecting peaks on a CF that better highlight the targeted signal properties. Phase arrivals are also picked on the CF to assign a change point t_i at the beginning of the i^{th} event.

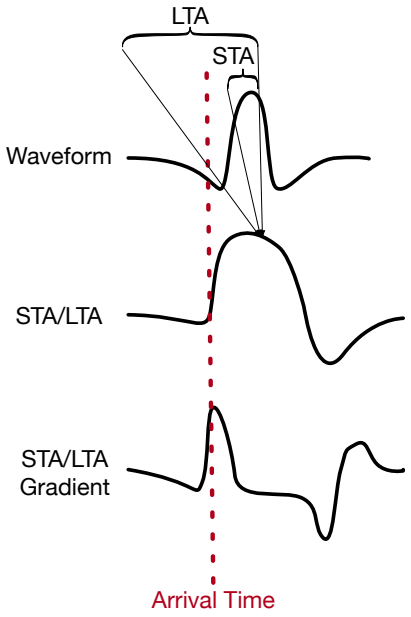


Figure 1.5: Demonstration of how to use STA/LTA as the CF to perform event detection and phase arrival-time picking on a Ricker wavelet.

Allen [1] proposed the CF in (1.1) for detecting seismic events based on a simple anomaly detection strategy. The characteristic function, $E(t)$, is

$$E(t) = f(t)^2 + C_2 \times f'(t)^2 \tag{1.1}$$

where $f'(t)$ is the first difference of the time sequence $f(t)$. The constant C_2 is used to vary the relative weight assigned to the amplitude and first difference, depending on the sampling rate and noise characteristics of individual seismic stations. The anomaly in this characteristic function is detected by thresholding the ratio of a short-term average of $E(t)$ to a long-term average, abbreviated STA/LTA, as summarized in Algorithm 1. Notice that there are already five parameters $(C_1, C_2, C_3, C_4, C_5)$ needed to be set for this part. The outputs of Algorithm 1 are passed into a quality control (QC) process to determine whether or not a detected anomaly is truly a seismic event, where even more parameters are required. Allen [66] later summarized the detection and picking scheme into two conceptual steps: (a) CF generation, and (b) anomaly (usually peaks) detection on the CF.

Algorithm 1: First part of STA/LTA algorithm proposed in Allen [1].

Input: Time samples $x_i, i = 1, 2, \ldots$

Output: short-term average α_i, long-term average β_i, and reference level γ_i

begin

- Store present value: $y_{i-1} \longleftarrow y_i$
- Filter to remove DC offset, producing updated value of Real data: $y_i = C_1 \times y_{i-1} + (x_i - x_{i-1})$
- Calculate weighted derivative: $\Delta y_i = C_2 \times (y_i - y_{i-1})$
- Store present value of time sample: $x_{i-1} \longleftarrow x_i$
- Compute CF: $E_i = y_i^2 + \Delta y_i^2$
- Compute short-term average of E: $\alpha_i = \alpha_{i-1} + C_3 \times (E_i - \alpha_{i-1}), \quad 0.2 < C_3 < 0.8$
- Compute long-term average of E: $\beta_i = \beta_{i-1} + C_4 \times (E_i - \beta_{i-1}), \quad 0.005 < C_4 < 0.05$
- Compute reference level: $\gamma_i = C_5 \times \beta_i$

end

To reduce the number of parameters in Allen [1], Baer and Kradolfer [67] replaced the constant C_2 in (1.1) by a time-varying value k_i defined as

$$k_i = \frac{\sum_{j=1}^{i} |x_j|}{\sum_{j=1}^{i} |x_j - x_{j-1}|} \tag{1.2}$$

This can be regarded as using the signal envelope as the CF. Indeed, when assuming that the signal x_i is narrowband, (1.2) reduces approximately to

$$k_i = \frac{1}{\omega_i \times \Delta t} \tag{1.3}$$

where ω is the instantaneous frequency and Δt is the sampling interval. Such a simplification makes (1.1) similar to the definition of an envelope function (EF): if k, which replaces C_2 in (1.1), is squared, CF becomes an EF. Thus, the new CF is defined as

$$E(t)^2 = f(t)^2 + \frac{f'(t)^2}{\omega_i(t)^2} \tag{1.4}$$

Since estimating $\omega_i(t)$ is slow in real time, Baer and Kradolfer [67] approximated k by k_i

defined in (1.2). Instead of viewing a seismic event as a time series, Morita and Hamaguchi [2] divided it into multiple locally stationary segments each modeled as an autoregressive (AR) process. Combining with the Akaike information criterion (AIC) [68], Sleeman and Eck [3] designed an AR-AIC method for single-component P-wave detection, which was later extended to three-component recordings by [69, 70, 71].

Higher-order statistics, including kurtosis [72] and skewness [73], have also been used to refine the picks due to their sensitivity to abrupt changes in a time series. These algorithms generally perform better for the P-waves than S-waves, most likely because S-wave arrivals are usually contaminated by the P coda and converted phases. Polarization has also been used to discriminate P and S phases [74]. The covariance matrix [75] is used to rotate waveforms into polarized P and S waveform components using methods such as the singular value decomposition (SVD) [76, 77]. Although each of these methods has seen success on one or more datasets, the most widely adopted automatic event detection and phase picking approaches remain the STA/LTA for event detection and the AR-AIC for phase picking. ObsPy, the standard seismic processing package in Python, includes a function named "ar_pick" to benchmark and automate this procedure.

As more and more seismic sensors are deployed in the field, array-based approaches have started to gain traction for seismograms recorded in noisy regions. To enhance the detection of weaker signals, array-processing techniques are often used, where a systematic delay-and-sum of signals from closely spaced receivers increases the signal-to-noise ratio (SNR) [79]. However, this strategy usually requires a set of precomputed travel times based on known velocity models that are not always available in many study regions. Cross-correlation of coherent signals from adjacent receivers, on the other hand, allows for the estimation of highly accurate relative delay times without a velocity model, from which we can estimate earthquake epicenter locations [80]. It relies on a set of predetermined event templates [81] and often results in new detections that are 5–10 times the number in the original catalog [82, 83]. These techniques are now referred to as matched filter

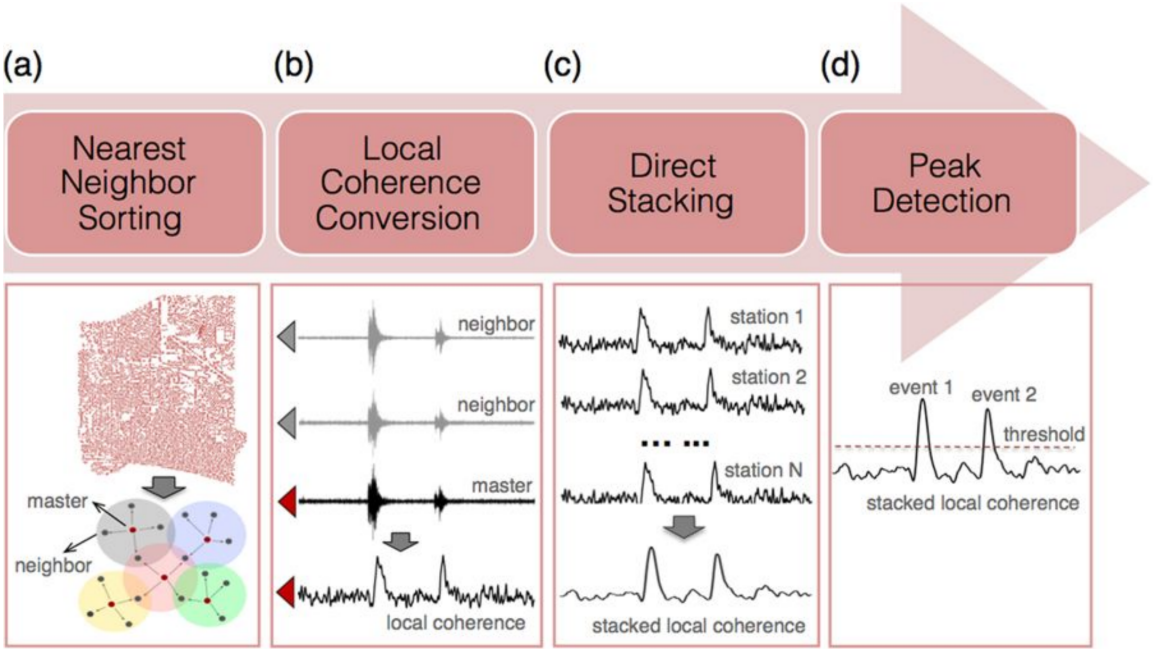


Figure 1.6: Processing pipeline for local similarity presented in Li *et al.* [78].

approaches. An auto-detection technique based on cross-correlation could be used to build templates from scratch [84]; however, the complexity of $\mathcal{O}(N^2)$ limits its scalability to large datasets, where N is the number of samples. Recent research [85, 86] proposes using PageRank [87] to speed up the search in a large template database. Fingerprinting, originally proposed by Baluja and Covell [88] for audio signals, is also used by Yoon *et al.* [89] to reduce the cost of computing waveform similarity and searching on a hashed table. However, none of these approaches address the $\mathcal{O}(N^2)$ complexity, rather they aim at reducing the cost of each computation step. Li *et al.* [78] proposed the "local similarity" measure for a dense surface array, where waveform similarity is only computed for adjacent receivers within a time window, to cap the complexity for large-N arrays within $\mathcal{O}(N)$. Zhu *et al.* [90] went on to formulate the phase picking problem as a binary classification between event picks and non-event picks to impose QC on the picked arrival times. While this greatly improves the detectability of weak events on a dense array, it is infeasible to apply this strategy for a coarse array with many fewer receivers. Nevertheless, the idea of classifying signals into event and non-event categories has facilitated the development of

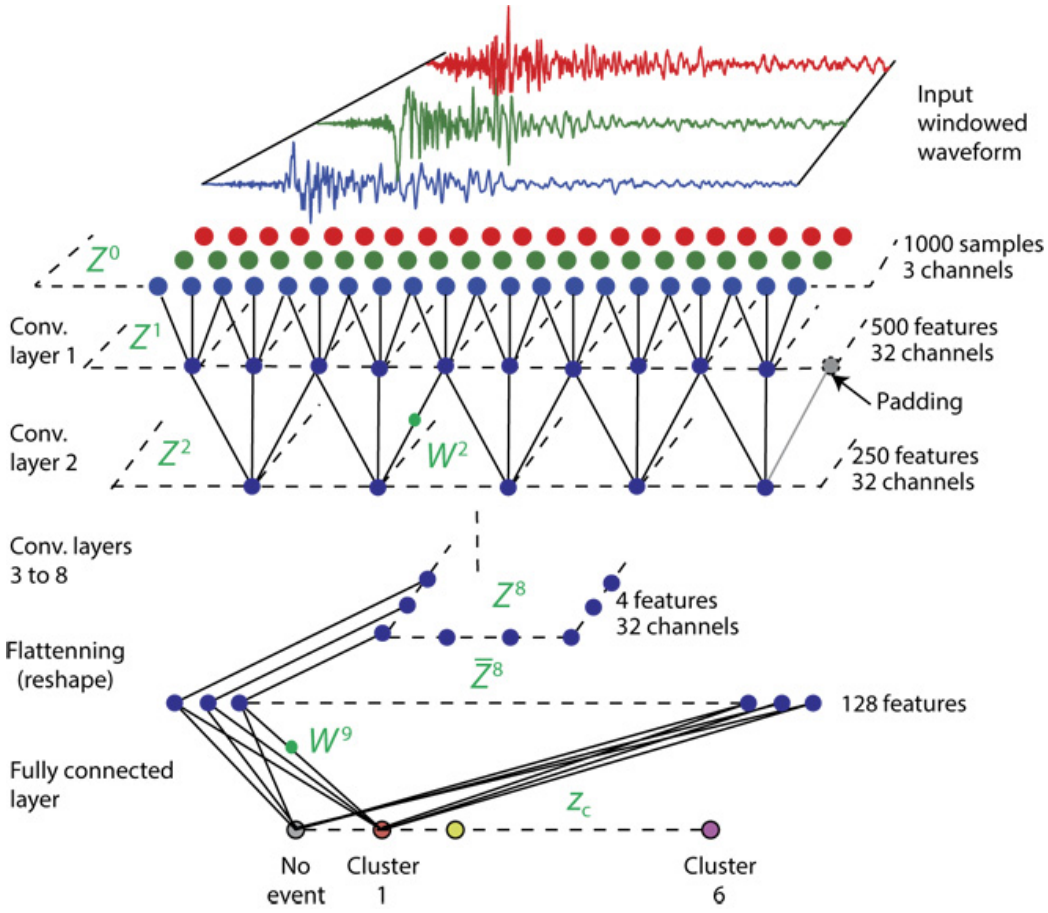


Figure 1.7: CNN structure used in Perol *et al.* [91] for seismic location categorization.

CNNs for event detection and phase picking [5].

Neural networks were applied to the seismic phase picking problem as early as the 1990s. McCormack *et al.* [92] proposed a NN trained with backpropagation on labeled trace edits and refraction picks. In 2018, shortly after Perol *et al.* [91] demonstrated CNN's success in classifying seismic waveforms into a finite set of regions of origin locations, CNNs quickly found their application [59] in other seismic problems, especially in phase picking [4, 6, 8]. Both Ross *et al.* [4] and Zhu and Beroza [6] assume a large labeled set trained on powerful GPU clusters. The trained model will be deployed on the retrieved continuous waveforms stored on a server on the cloud. Zhu *et al.* [8] investigated the possibility of training a simplified CNN on smaller datasets with a limited number of labeled samples. This is more suitable for small or new study regions where existing picked arrival

times are not plentiful. However, all existing studies have focused on training and deploying CNN on cloud-only setups. The low-frequency of data retrieval from seismic sensors and the high cost of wireless transmission in the field make it nearly impossible to deploy those CNN models for a real-time system on a large scale.

1.2 Future is on the Edge

Traditionally, seismic sensors are deployed in the field as battery-powered devices and the data is not retrieved until weeks or months after the first deployment. During data retrieval, both the battery and hard disk are swapped to maintain continuous recording. This acquisition scheme, shown in Figure 1.8, is reliable and consumes minimum battery energy since there is no data transmission and no local computation. Thus, it is still the most popular deployment in the field. However, as the number of seismic sensors increases, manual retrieval of the waveform data becomes tedious and hard to manage by a small group of technicians. Moreover, it is not possible to have real-time monitoring for this traditional acquisition scheme as the data is not accessible until months after being recorded.

With advances in battery technology, transferring raw waveform data has become possible using GSM networks. This enables semi-real-time processing of the seismic waveforms, including neural networks, on servers in the cloud, which is referred to as the cloud computing approach [93]. Indeed, the trained NNs can take the enormous number of labeled samples and iteratively find the optimal model to achieve the best accuracy. Moreover, the abundant computing resource, including multiple-GPUs in the cloud, can lead to nearly instantaneous inference speed, in which the arrival times in a waveform are picked in a fraction of second. However, as more seismic sensors are deployed in larger monitoring regions, the cost of hundreds or thousands of GSM transmitters and monthly subscription fees can add up quickly. With the improved recording quality of seismic waveforms, the transmission of full three-component waveforms will require more transmission bandwidth. The bandwidth problem is alleviated by running an event detection program, e.g.,

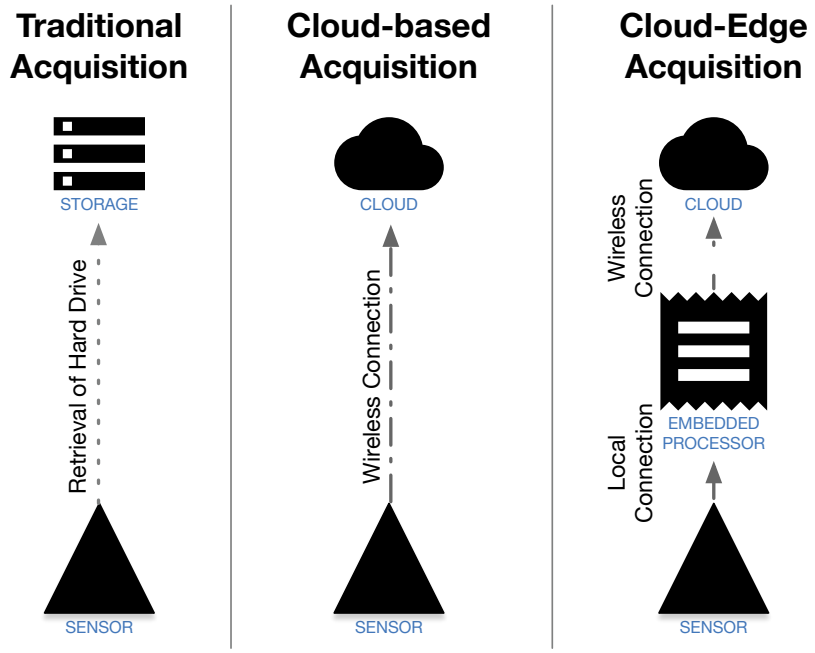


Figure 1.8: Comparison of three acquisition schemes: (a) no computing or communication at the sensor, (b) cloud computing after wireless transmission, and (c) computing at the sensor and in the cloud.

STA/LTA, on the sensors locally, which is referred to as cloud-edge acquisition in Figure 1.8. Now, only the waveform segments with a detected seismic event, also known as triggered waveforms, are transmitted to the cloud for real-time processing. Nevertheless, the energy cost for transmitting segments of full waveforms to the cloud puts pressure on the overall energy consumption and may result in more frequent battery replacement in the field. Even if the transmission problem can be mitigated by the new 5G network and improved batteries, the latency introduced when to transmitting data to and from the server significantly limits the usage of such a real-time system. In addition, privacy and data security concerns are often raised about transmitting the full waveform to a third-party server, since the full waveforms may be used to invert sensitive information that is not necessary for later processing but is harmful to clients should it be leaked [94]. As shown in Figure 1.4, if the arrival times can be picked accurately on the embedded processor locally on the seismic sensor, only the arrival times need be transmitted to the cloud. In this way, both transmission and privacy issues can be resolved.

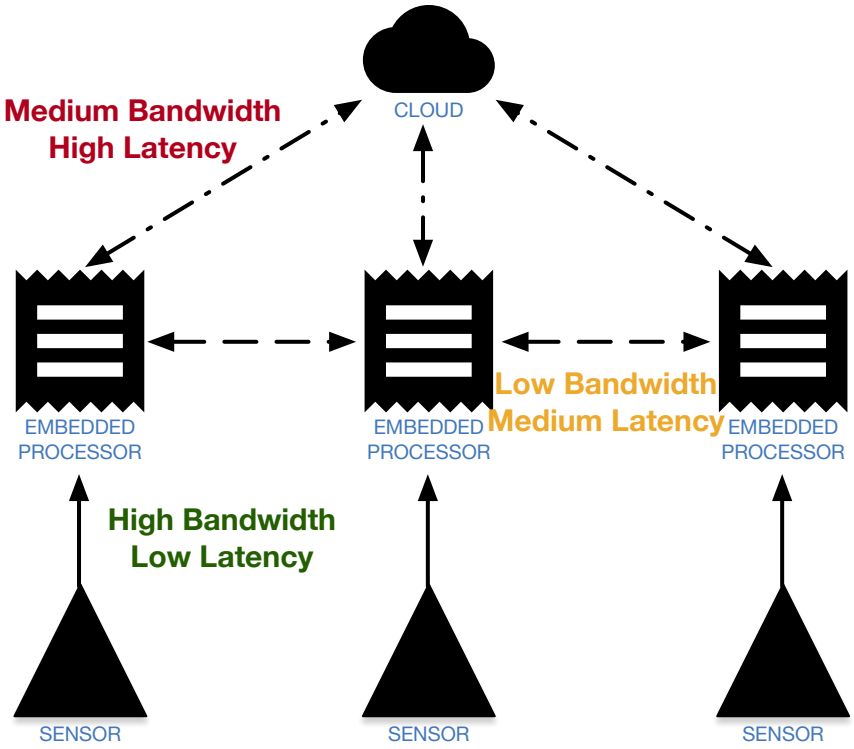


Figure 1.9: Communication comparison between cloud server and edge devices.

Edge computing [95] was born acknowledging the importance of proximity for systems such as the internet of things (IoT). In contrast to cloud computing, edge computing favors the processing of data where it is created [96], as shown in Figure 1.9. The benefits are clear: transmission is only necessary for the smaller processed data, which is beneficial for end-to-end latency, system scalability, and data privacy. Processed data is usually smaller in size so transmitting it takes less time and energy. Given a fixed bandwidth, more end devices can be added to a network. Should a malicious third-party get access to the data pipeline, it does not have access to the raw data which might contain valuable information that is not necessary for the computing system. However, the challenges are also obvious. Both the computation power and data access are limited on the edge device. The embedded processor used in IoT is usually clocked at a lower frequency and has fewer cores. Instead of having an aggregated dataset with waveforms from all receivers, we now only have access to those on a single station. Thus, two problems need to be solved before prototyping

a feasible solution for applying NN models to the seismic phase picking problem, taking advantage of both cloud and edge computing.

Based on the acquisition schemes in Figure 1.8 and communication models in Figure 1.9, a cloud-edge processing schemes similar to Sepulveda and Pulliam [97] is proposed as shown in Figure 1.10. In addition to the IoT system of Raspberry Pi and seismometers,

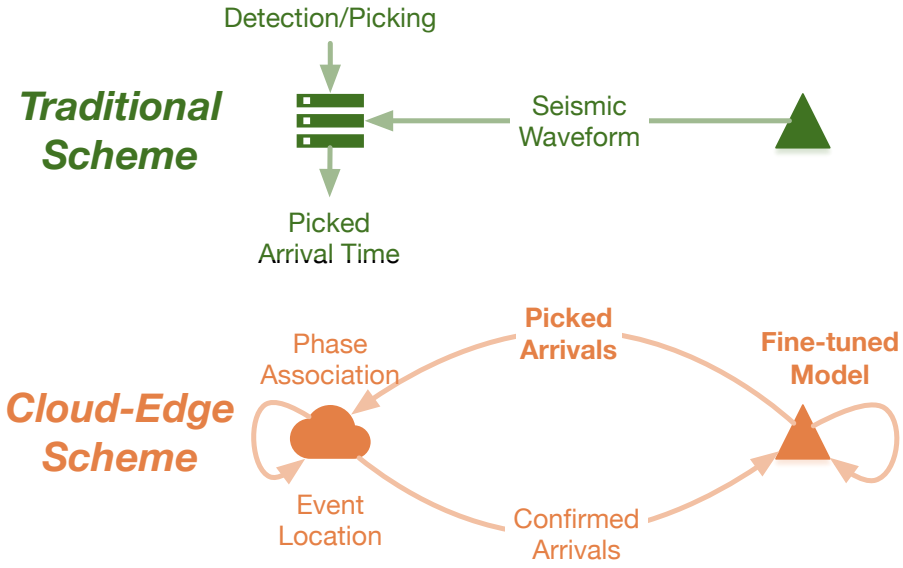


Figure 1.10: Traditional processing scheme (top) and proposed cloud-edge processing scheme (bottom). Communication in the traditional scheme is one way from sensor to server; that in the cloud-edge scheme is a closed loop.

the proposed scheme included NN models that can be automatically fine-tuned and improved over the communication between cloud server and edge devices. The traditional processing scheme separates the acquisition device from the processing device which results in one-way communication as shown in Figure 1.10. The seismic waveforms are either manually retrieved from hard disks or transmitted via GSM networks to the central processing computer. Phase arrival times are then determined by manual picking from seismology experts, automatic phase picking programs, or event neural network models based on the full waveform information. The proposed new scheme takes advantage of the extraordinarily accurate phase picking by the neural network models [4, 6, 8] and deploys those models in the edge devices. In this case, the data transmitted from the edge

device to the cloud server is reduced to individual arrival times from each sensor which is much smaller in size compared to full waveforms, and transmitted less frequently than the continuous waveform data. The arrival times from different sensors are aggregated in the cloud server via phase association, done as part of the event location program. Then, arrival times associated with a seismic event are confirmed as valid phase arrivals. Thanks to the added two-way communication capability between cloud servers and the edge devices, the confirmed arrival times can be fed back to the corresponding edge devices. The confirmed arrival times might then used to fine-tune the NN models and improve the detection and picking accuracy. This closed computing loop in Figure 1.10 automatically improves its sensitivity to the targeted seismic phase arrivals and gradually adapts itself to changing acquisition conditions. With better phase association and event location programs on the cloud server, the improvements are passed down to the edge devices with a small communication price between cloud and edge and used for some fine-tuning on the edge devices at a small computation cost. With transfer learning validated for seismic phase picking [98], the proposed system can start from a general initial model trained for one region, and gradually adapt to characteristics of a new monitoring area.

1.3 Roads Ahead

Two major steps needed for deploying NN models for seismic event detection and phase picking are training and inference. While the training of neural networks is better done on a powerful GPU cluster in the cloud, inference may be more favorable on the edge as data is acquired. This thesis lays down the groundwork of the proposed computing scheme shown in Figure 1.10 by investigating two problems: (a) design a NN model that can be trained on small to medium size sampled dataset; (b) prepare this pre-trained model into a feasible candidate for deployment on the target edge device. The work builds on the traditional acquisition and processing schemes shown in Figure 1.11. The continuous seismic waveforms are acquired and processed into waveform segments via traditional schemes.

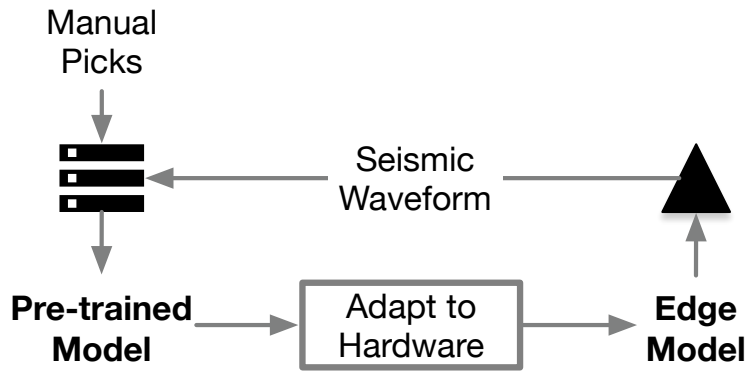


Figure 1.11: Flow diagram of adapting a traditional processing system to the proposed scheme. Neural network models are trained on the acquired seismic waveform and historical picks. The pre-trained model is then simplified and quantized to an edge model ready for deployment on the edge devices.

The resulting manual picks are used as the labels of waveform segments for training the neural network model. The pre-trained model is then modified to an edge model adapted to the hardware conditions and eventually deployed on the target edge devices.

Chapter 2 demonstrates a feasible design for an event detection and phase picking system based on a neural network model, named CNN-based phase identification classifier (CPIC). Specifically, the CPIC model can be trained on small to medium size labeled datasets with reasonably high classification accuracy. The complete system is validated on the continuous waveform for event detection and phase picking whose results are considerably better than the traditional method. Moreover, the pre-trained model can be transferred to different regions where the number of manually picked phases are limited. In addition, CPIC is helpful in significantly expanding the seismic catalog with reliable detection and accurate pickings of previously unlabeled events.

Given a pre-trained NN model, Chapter 3 discusses deployment details when there are limited computing resources, such as on embedded devices. The pre-trained model is first simplified based on the hardware implementation conditions to reduce the number of overall trainable parameters in the model. Since the seismic waveform is originally recorded with fixed-point analog-to-digital converters (ADCs), quantization of arithmetic

operations in the model is also considered to reduce the numerical precision by a finite wordlength system and speed up matrix-matrix multiplication operations. Conclusions are made in Chapter 4 summarizing the contribution of this thesis. Potential extensions and improvements discussed as future work include fine-tuning performed on the edge devices and a distributed computing scheme for multiple edge devices.

CHAPTER 2

DESIGN NEURAL NETWORK WITH LIMITED RESOURCE

Many recent studies about seismic event detection and phase picking converge to apply neural networks on large training sets to improve the detectability [4] and picking accuracy [6]. While it is beneficial to have sophisticated neural network models trained on large training sets, neither of these two works validate their performance on small to medium training sets where the number of labeled samples are limited in the training set. Large training sets with millions of labeled samples are only available for large well-studied regions with an extended observation history, such as California, U.S.; usually, only a few thousands of labeled samples are available for small or new regions. The waveforms from these regions tend to be actively processed by a small group of researchers; thus, a reliable automatic event detection and phase picking algorithm are crucial to the success and efficiency of such tasks. Light-weighted neural network solutions are first investigated by Zhu *et al.* [99] in pursuit of a small model that can be trained stably on a small set with a few thousands of labeled samples.

In this chapter, the CPIC model proposed in Zhu *et al.* [8] is explained in details as the solution of getting a pre-trained model in Figure 1.11. Section 2.1 described the overall design of CPIC system as well as the core CNN classifier. Five datasets used to train and validate the CPIC model is explained in details in Section 2.2. In Section 2.3, a unique preprocessing technique is designed to facilitate the earthquake recordings as well as future deployment on embedded systems. Post-processing is explained in Section 2.4 to detect seismic events and pick phase arrival times on the CNN outputs from a moving window of waveforms. Training of the CPIC model was initially conducted on the medium-sized Wenchuan dataset and validated for both detection and picking on the continuous waveform in Section 2.5. It is later expanded to both small and large datasets in Section 2.6 to verify

the potential of over-fitting problem. Real-world examples are given to demonstrate the usage of the resulting CPIC model with transfer learning. Comparison with other neural network based works is given in Section 2.7, which explains the advantage of CPIC model for deployment on embedded devices.

2.1 CNN-based Phase Identification Classifier

2.1.1 Processing Pipeline

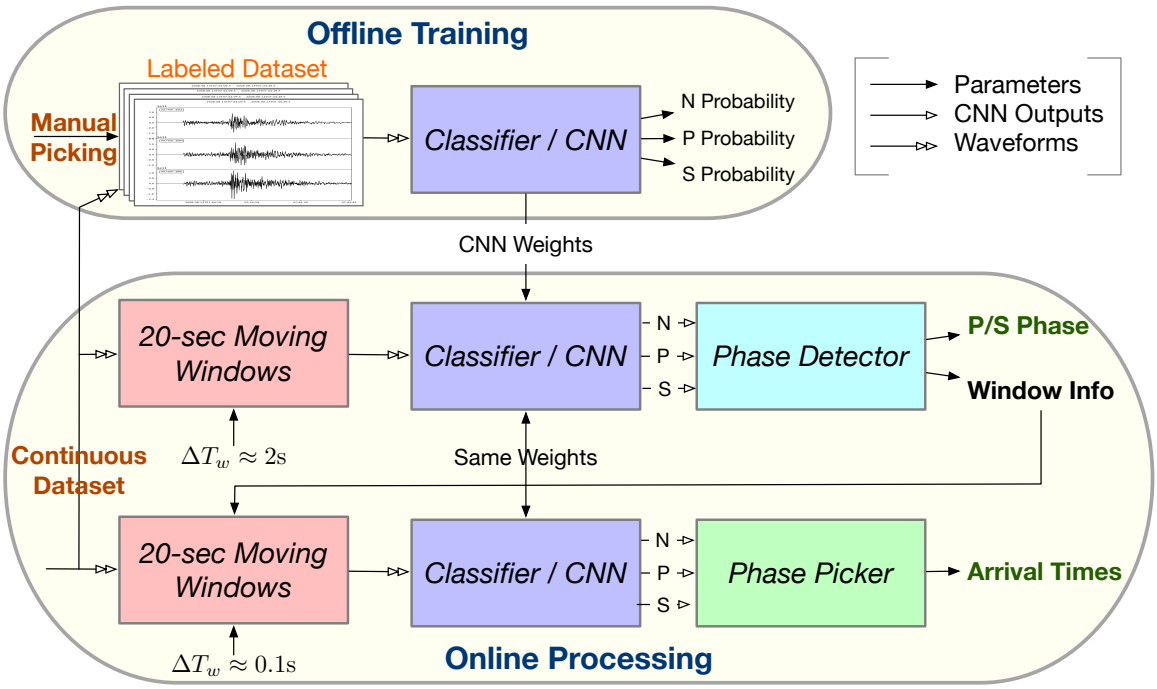


Figure 2.1: CPIC flow chart. Inputs are three-component seismograms recorded at a single station, shown in red boxes. Detector outputs are P-wave, S-wave or noise window probabilities, shown in the cyan box. Detected windows a reprocessed with very fine offsets controlled by $\Delta T_w \approx 0.1$ s to obtain picked arrival times for P and S phases, which are shown in the green box.

The overall processing pipeline of CPIC system proposed in Zhu *et al.* [8] is summarized in Figure 2.1. An off-line training process optimizes the parameters of the *CNN-based classifier* iteratively over the labeled dataset. The trained classifier is then used during on-line processing for both phase detection and picking. The *Phase detector* employs moving

windows with 90% overlap ($\Delta T_w = 2$ s offset) and casts seismic phase detection as a classification problem of P-wave, S-wave, or noise-only labels. The detected windows are then reprocessed by the same classifier to generate characteristic functions (CFs) on a finely sampled grid, e.g., $\Delta T_w = 0.1$ s offset. The *phase picker* estimates the arrival times based on the peaks of smoothed CFs. Multiple window offsets, ΔT_w, were tested in a grid search manner. In general, a smaller ΔT_w gives better picking accuracy; however, the computation cost is also inversely proportional to ΔT_w.

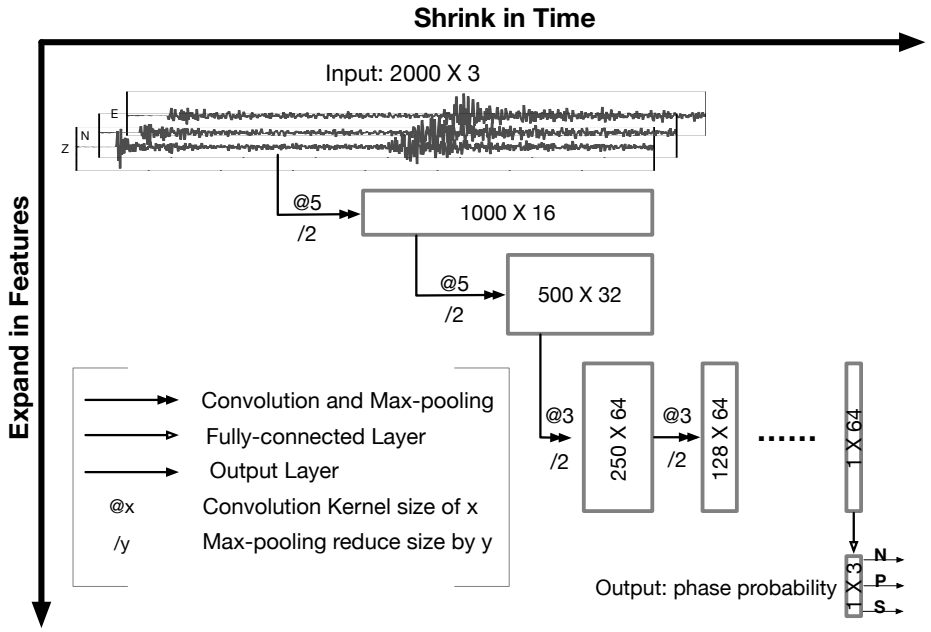


Figure 2.2: A diagram showing the CNN network structure. Each input is a 3-C seismogram (20-s window) which shrinks in time but expands in the feature dimension as it passes through 11 convolutional layers for feature extraction. The final layer is fully connected with 3 outputs that give the probabilities of a window being noise, P, and S phases.

Unlike other CNN based approaches [6, 100], the design here separates the generation of CFs from the actual picking process. This may increase the number of user parameters needed to set, such as ΔT_w; however, this design also allows great flexibility in applying the trained model by decoupling CF generation from peak detection. Given a seismic waveform with a fixed sampling rate, the phase picker can determine dynamically the picking resolution in real time depending on the system resources available at that instant. Since

the computation cost of such moving window design is linearly proportional to the time resolution, a reasonable tradeoff can be made when computation resources are limited on the battery-powered devices.

2.1.2 CPIC model

The core of the system in Figure 2.1 is the CNN-based classifier. It operates on inputs that are 3-C seismograms in 20-s windows, sampled at 100 Hz. Its three outputs are probabilities of each window containing a P/S phase arrival at 5 s, or only noise. The CNN classifier contains 11 convolutional layers followed by one fully-connected layer (Figure 2.2). Between each layer, a rectified linear unit (ReLU) activation function [54] is added to introduces nonlinearity into the model. Max-pooling layers [101] are used for downsampling the time domain at the output of each layer. To accommodate small to medium training set sizes, the proposed CNN model uses only one convolutional layer (CONV layer) between each max-pooling layer. Multi-path CONV layers, such as those used in Inception V4 [64], were also considered; however, poor convergence on a small training set makes this a less favorable structure. Batch normalization [58] is also integrated to further stabilize the training process. The proposed CNN model is then trained using Adam [102] by processing many labeled seismic waveforms known to contain P or S phases, or noise only.

The outputs of the final fully-connected layer (FC layer) are three real numbers representing the probabilities of the 20-s input waveform window belonging to the corresponding classes. A *Softmax* function is used to normalize these probabilities in the output layer:

$$q_i(x) = e^{z_i(x)} / (e^{z_0(x)} + e^{z_1(x)} + e^{z_2(x)}) \tag{2.1}$$

where $i = 0, 1, 2$ represents noise, P, and S classes, and $z_i(x)$ is the unnormalized output of the last FC layer layer for the i^{th} class. A loss function is needed when optimizing the CNN weights during the training process, so we use the cross-entropy between a true probability

distribution p and the estimated distribution q which is defined as

$$H(p, q) = -\sum_{x} p(x) \log q(x) \tag{2.2}$$

Hence, the *Softmax* classifier minimizes the cross-entropy between the estimated class probabilities (q defined in (2.1)) and the "true" distribution, which is the distribution where all probability mass is on the correct class, e.g., $p = (0, 1, 0)$ for a labeled P phase window.

The resulting CNN model in Zhu *et al.* [8] has 107,248 parameters when trained on 20-s windows of a 3-C seismic waveform. The number of parameters can be reduced if a shorter window length is chosen instead. Since each layer down-samples the input data by a factor of two, the model can adjust to a different window length by adding or removing layers. Finally, the number of FC layers used here is fewer than commonly seen in CNNs. Experimentation with different numbers of FC layers (one, two, and three) found no discernible difference in the classifier accuracy. Indeed, the last several CONV layers seem to serve a similar role as regular FC layers without introducing as many parameters. For example, the number of parameters used in three FC layers layers to go from layer eight to the output is 69,728, while using the proposed CONV layers only takes 37,056 parameters. The balance point where these two approaches result in a similar number of parameters is at layer 10, where the final CONV layer layer only has one output in the time domain. For the sake of simplicity, the current model with only one FC layer was chosen.

2.2 Datasets and Study Regions

Well studied regions such as California, U.S. have accumulated a large collection of picked arrival times for historical recordings. These extensive catalogs enable research like Ross *et al.* [4] and Zhu and Beroza [6] to look at datasets with millions of labeled samples and train relatively large CNN models. Ross *et al.* [4] made their huge training dataset available for public testing, which contains over 4.7 million labeled phases. Unfortunately, the entire raw

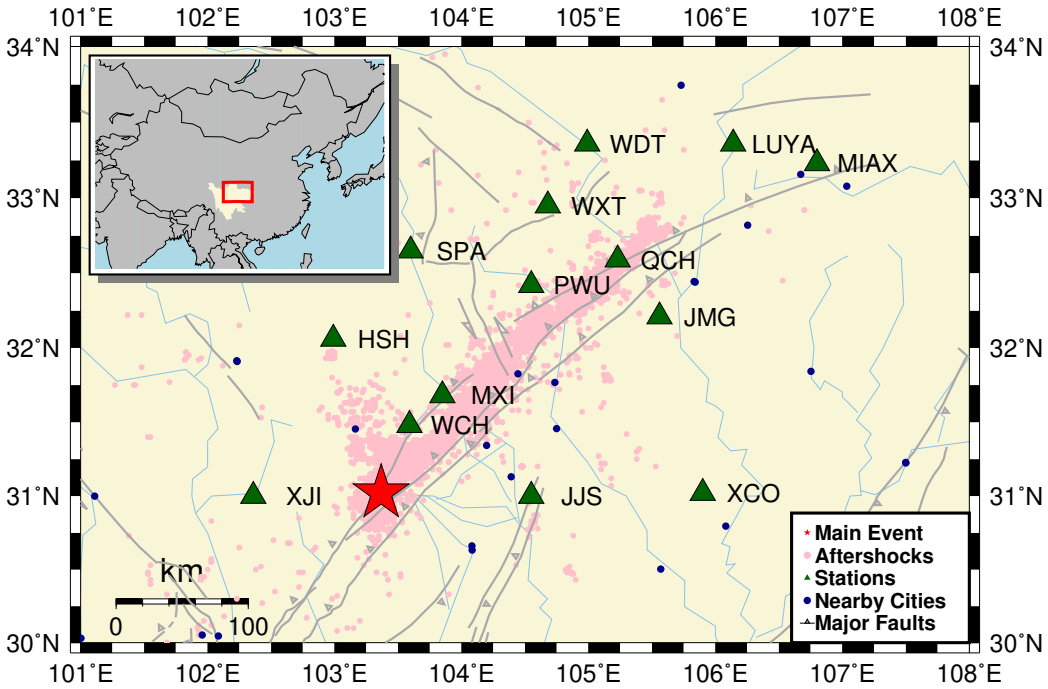


Figure 2.3: Map showing the study region in Sichuan, China along the aftershock zone of the 2008 M_W 7.9 Wenchuan earthquake (red star). The 9,361 manually picked aftershocks are marked as pink dots. The green triangles mark the 14 permanent stations that were used in this study. The gray and thin blue lines mark active faults and rivers in this region.

dataset has not been published; instead, a limited version is available, but with bandpass filtering and only 4-s three-component windows. This would restrict testing of the CPIC model by imposing a different, and undesirable, pre-processing scheme. CPIC has been validated on this dataset after modification adapting to the 4-s windows.

In Summer 2017, a big data competition in China Fang *et al.* [103] provided a unique opportunity to explore the feasibility of training NN on a small to moderate sized dataset. The CPIC model is trained on this earthquake aftershocks dataset which consists of approximately 30,000 labeled samples. Although relatively small, this dataset represents a typical scenario when analyzing the aftershocks of a major earthquake: existing algorithms or analysts can easily pick strong aftershocks at a later time; however, the real targets are the numerous aftershocks occurring right after the mainshock which are often missed by traditional methods [104, 105]. Another tiny aftershock dataset was available to us near the Mariana Trench for validating training. Six stations are recording continuously, and 2,004

Table 2.1: Summary of dataset used to train and test the CPIC model.

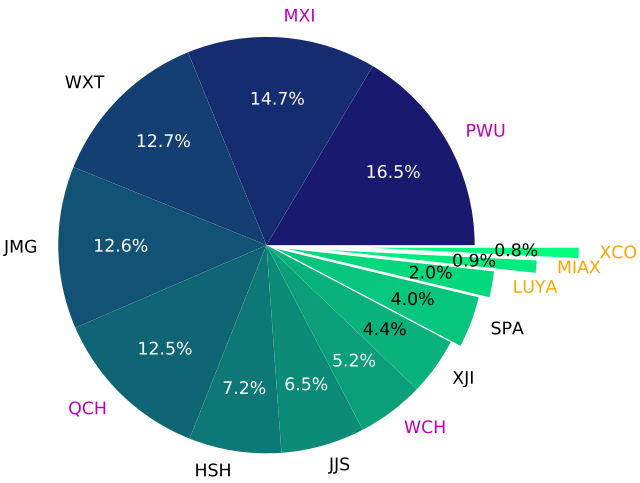
Dataset Names	Short Names	Train	Test
2008 M_W 7.9 Wenchuan Earthquake aftershock dataset	Wenchuan	YES	YES
Southern California Seismic Network phase dataset	SCSN	YES	NO
Mariana deep sea dataset	Mariana	YES	NO
Oklahoma induced seismicity dataset	OK	NO	YES
2019 M_W 7.6 Kokopo Earthquake aftershock dataset	Kokopo	NO	YES

local earthquakes were confirmed by seismic relocation method [106] with overall 18,354 labeled samples. Last but not least, the generality of CPIC was tested on an induced seismic dataset from Oklahoma, U.S. with three stations, as well as an aftershock dataset with a single station in southern Pacific ocean. The datasets used for training and validation of CPIC are summarized in Table 2.1 and their details are given in this section.

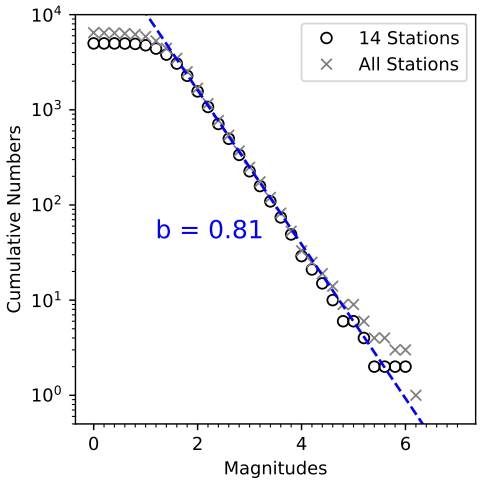
2.2.1 2008 M_w 7.9 Wenchuan Earthquake Aftershock Dataset

CPIC [8] was originally trained on the aftershock dataset of the 2008/05/12 M_W 7.9 Wenchuan earthquake, which was made public during a recent competition for identifying seismic phases [103]. Zhou *et al.* [100] is a similar work on a dataset of the same region using both CNN and RNN for event detection and phase picking tasks. Note that the dataset used in Zhu *et al.* [8] is a subset of that in Zhou *et al.* [100], which contains more stations monitoring a more extended period.

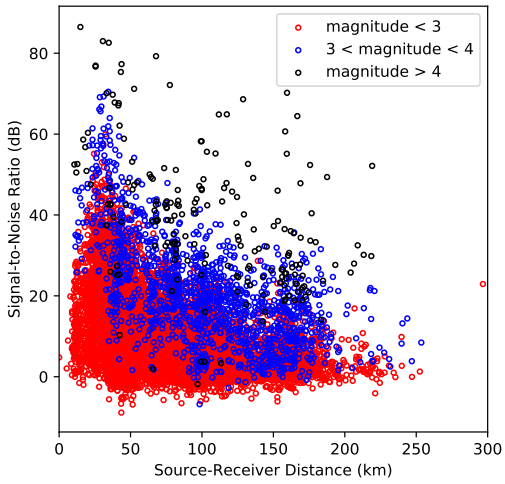
Study region The mainshock occurred on the eastern margin of the Tibetan Plateau (Figure 2.3), and ruptured the central and northern section of the Longmenshan fault zone [107, 108, 109]. Numerous aftershocks occurred following the mainshock, but many of them were still missing in any published earthquake catalogs [110]. The aftershock dataset includes continuous data recorded for one month by 14 permanent stations in August 2008, which is three months after the Wenchuan mainshock. Figure 2.4a shows the distribution of those phases among the 14 stations. Stations near the aftershocks and the rupture zones



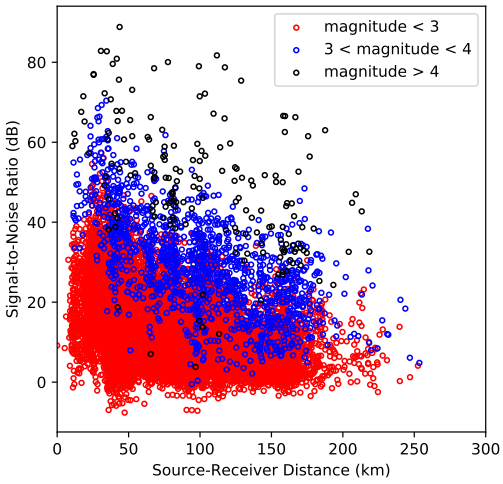
(a) Event distribution over stations



(b) Event distribution over magnitudes



(c) P phases



(d) S phases

Figure 2.4: Distribution of catalog events in the Wenchuan aftershock dataset for (a) different stations and (b) different magnitudes. Stations on or close to the rupture zone are marked in purple while those far away are marked in gold. Signal-to-noise ratio of picked arrivals against event magnitudes and source-receiver distance for (c) P phases and (d) S phases.

(e.g., PWU, MXI, WXT, JMG, and QCH) had most of the picked phases, while distant stations (e.g., XCO, MIAX, LUYA, and SPA) have very few; and station WDT has no catalog phase arrivals.

Catalogs The catalog we used contains 4,986 events with 30,146 phases manually picked on 14 permanent stations with arrivals of P (15,185) of S (14,961) phases. Figure 2.4b shows the catalog events distributed versus magnitude between M_L 0.3 to M_L 6.2. The SNR of each phase is computed as the ratio of signal powers between two 4-s waveforms: one after each phase pick (signal) and one before its corresponding P arrival (noise). Figures 2.4c and 2.4d show the distribution of SNR of P and S phases against event magnitudes and source-receiver distance.

Labeled dataset The training is conducted on a dataset of labeled seismic waveforms in 20-s time windows. A long time window was chosen so that there is a high likelihood that a P-wave window contains some S-waves at its end and that S-wave windows contain some P-wave coda at the beginning. This window definition implicitly embeds the normal sequential relationship between P and S wave phases in the labeled dataset itself. Adding noise-only windows, which are not included in the original labeled dataset, improves model performance against noisy seismograms. Here, we assume that quiet regions exist between 60 s after an S-wave phase and 60 s before a P-wave phase and generate 30,130 noise-only windows (red windows in Figure 2.5). We note that because those noise windows were not verified manually, it is possible that they may include small aftershocks not listed in the catalog. In the end, we obtain a dataset with 60,276 labeled windows, for which P-wave and S-wave or noise labels have been assigned.

Continuous Dataset Once the model is trained on the labeled dataset, the phase detector and arrival picker are then tested on the entire one-month continuous waveforms starting on 08/01/2008 00:00:00 Beijing Time (or 07/31/2008 16:00:00 UTC). Due to challenging acquisition conditions in the study area, there are some gaps in the continuous recording. These are filled with zeros to keep the overall dataset consistent while avoiding false detections.

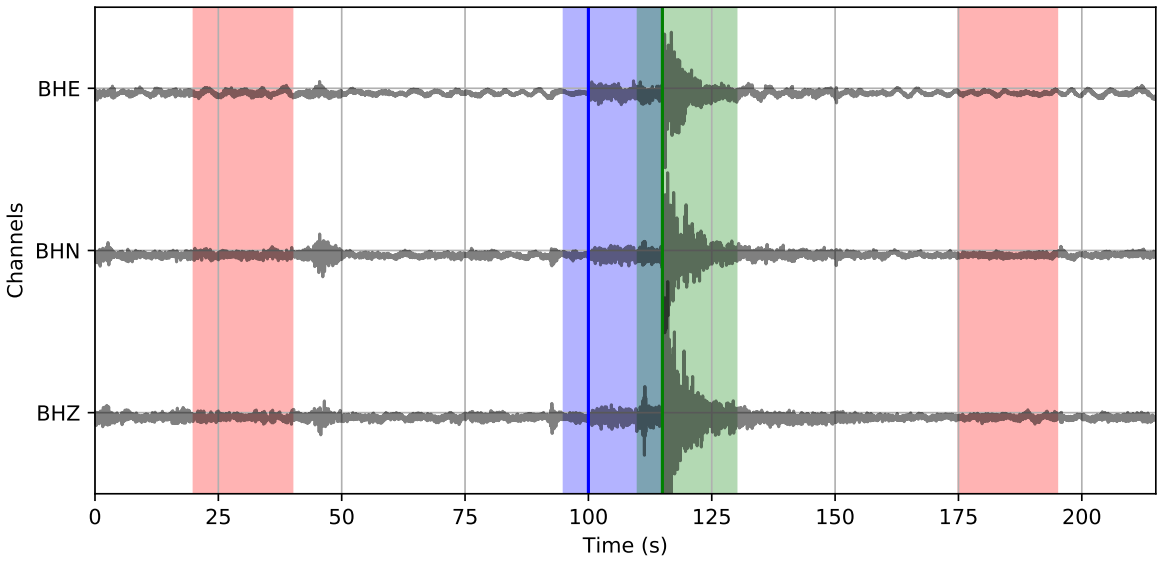


Figure 2.5: An example of three-component seismogram recorded at station HSH from which 20-s time windows are extracted for both P (blue) and S (green) phases. Noise (red) windows are cut one-minute before P and after S phases. Sampling rate is 100 Hz. The arrival times of P and S phases are marked by vertical blue and green solid lines, respectively.

2.2.2 Additional Datasets for CPIC Training

The 2008 M_w 7.9 Wenchuan Earthquake aftershock dataset is of medium size with about 60,000 labeled samples. To study the generality and stability of the proposed CNN classifier, two more datasets are studied, namely the SCSN dataset and the Mariana dataset.

The SCSN dataset was made available after the publication of Ross *et al.* [4]. A total of 4.7 million three-component seismic records were used for training and validation of the generalized phase detection framework. The data first were detrended and high-pass filtered above 2 Hz to remove microseismic noise, and all data were resampled at 100 Hz. Strong-motion records were integrated into velocity. These three-component records consist of 1.6 million P-wave seismograms, 1.6 million S-wave seismograms, and 1.6 million noise windows, with each being exactly 4 s (400 samples) in duration. The magnitude range of the data was between M_L -0.81 and M_L 5.7, while only records with epicentral distances less than 100 km were used. P-wave and S-wave windows were centered on the respective analyst pick, while noise windows were defined starting 5 s before each P-wave pick. These

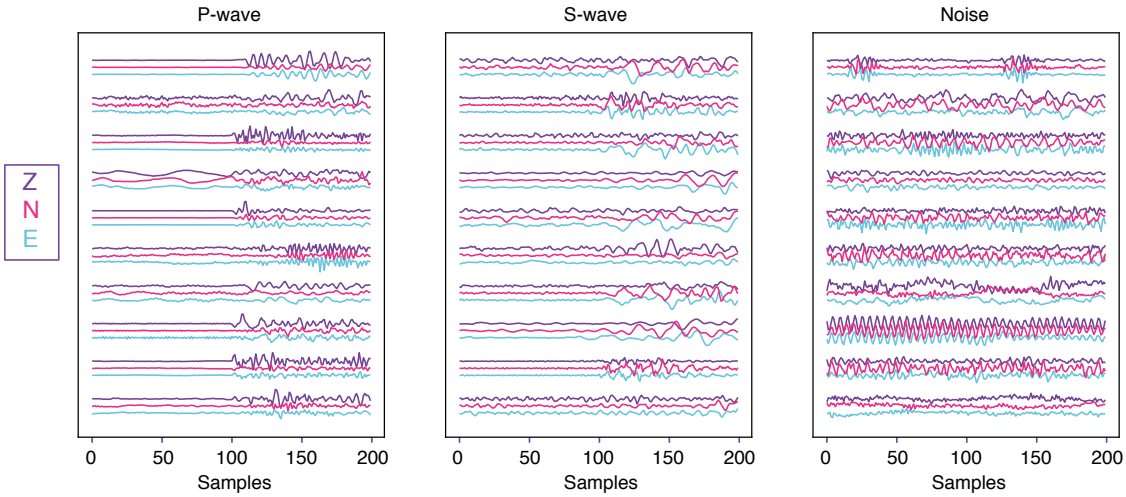


Figure 2.6: Example of the first 2 s seismic waveforms from SCSN dataset for P-wave, S-wave, and noise traces. They were given in Figure S2 by Ross *et al.* [4].

windows form the set of features used as input to the CNN, and examples of each class can be seen in Figure 2.6. The even distribution of records between each class ensures that the training process is not biased towards any one class. Then, each three-component record was normalized by the absolute maximum amplitude observed on any of the three components. No other pre-processing was performed on the datasets.

The Mariana dataset is the smallest dataset among all three of the training sets. Shown in Figure 2.7, six stations were deployed on one side of the trench (displayed as the dark blue region) while the main shock took place on the other side. Numerous aftershocks happened after the mainshock, but there were only 53 local earthquakes identified in the catalog. Using the matched filter technique discussed in Section 1.1.2, 7,634 local earthquakes were detected. Among them, 2,004 events are confirmed by the seismic relocation algorithm [106]. The training set consists of P-wave and S-wave waveforms from these confirmed events as well as noise waveforms cut around the detected phases. The overall number of labeled samples is only 18,354, half of which are noise waveforms. Notice that, unlike the event in SCSN dataset that has a wider range of magnitude distribution, the events in the Mariana dataset resembles those in Wenchuan dataset that CPIC was originally trained on.

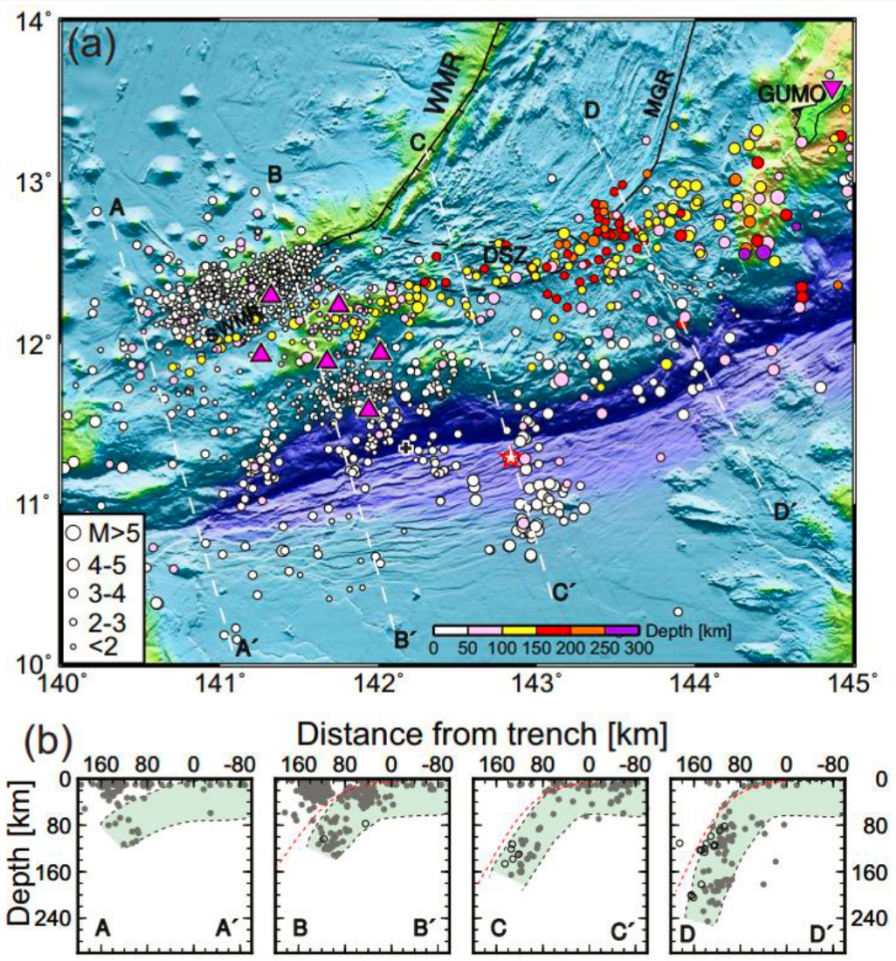


Figure 2.7: Study region of the Mariana dataset: (a) Map around the Mariana trench with six stations marked as pink triangles. The main event is marked by the red star. (b) Four vertical sections are taken along the white lines to demonstrate the subduction zone around the trench. It is taken from Figure 9 in Zhu *et al.* [111].

2.2.3 Additional Datasets for CPIC Validation

To validate how well CPIC generalizes to other datasets, we apply the CNN trained on aftershocks in Wenchuan, China to a dataset containing likely human-induced earthquakes in Oklahoma (OK), USA [112]. As shown in Figure 2.8, 890 events were manually picked with P and S phases on three stations (OK025, OK029, and OK030) from the local catalog. Noise waveforms are taken one minute before P-waves and one minute after S-waves. This results in a small catalog dataset with approximately 5,000 labeled samples. Notice that source-receiver distance of the three stations is clearly different in this dataset. Unlike the

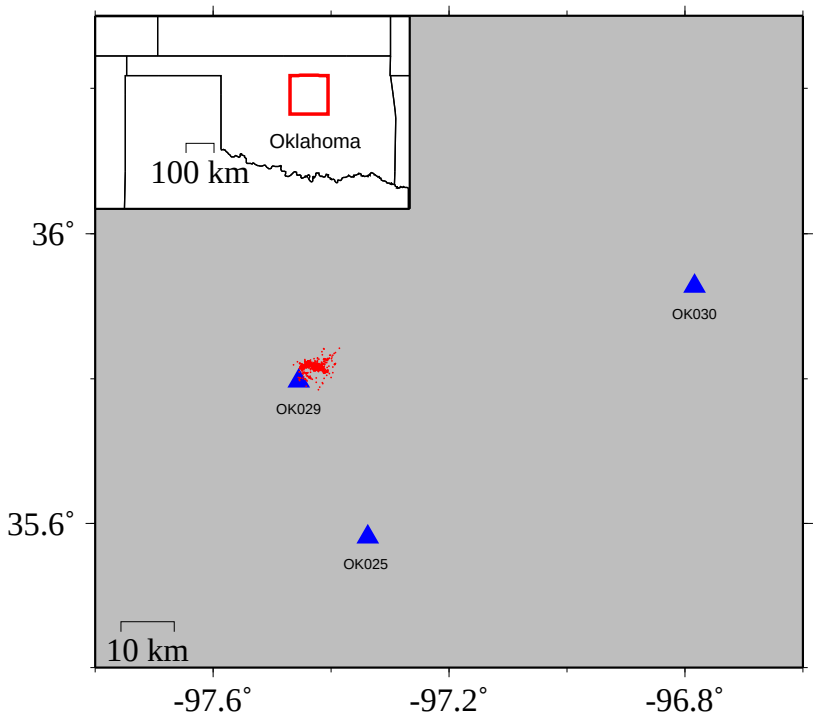


Figure 2.8: Map of study region in Oklahoma, central U.S. Red dots are 890 events with P and S phase arrivals and blue triangles are broadband stations of the United States Geological Survey Network (USGS).

aftershocks shown in Figure 2.3, induced events are concentrated around station OK029.

On 05/14/2019, a magnitude 7.5 earthquake occurred 45 km NE of Kokopo, Papua New Guinea at a depth of 10 km. The Solomon and Vanuatu Islands are subduction-related features caused by the subduction of the Australian Plate beneath the greater Pacific Plate. It is a seismically active area having frequent large earthquakes. The earthquakes are caused by the northeasterly movement of the Australian Plate as it dives beneath the Pacific Plate, but there are variations along the plate boundary. Numerous aftershocks occurred in the first few hours, as shown in Figure 2.10b, but only two are listed on the USGS website as shown in Figure 2.10a.

2.3 Pre-processing

Prepossessing steps vary from method to method and dataset to dataset, but it is possible to identify two general trends. The first group of studies [114, 4, 115] try to remove as much

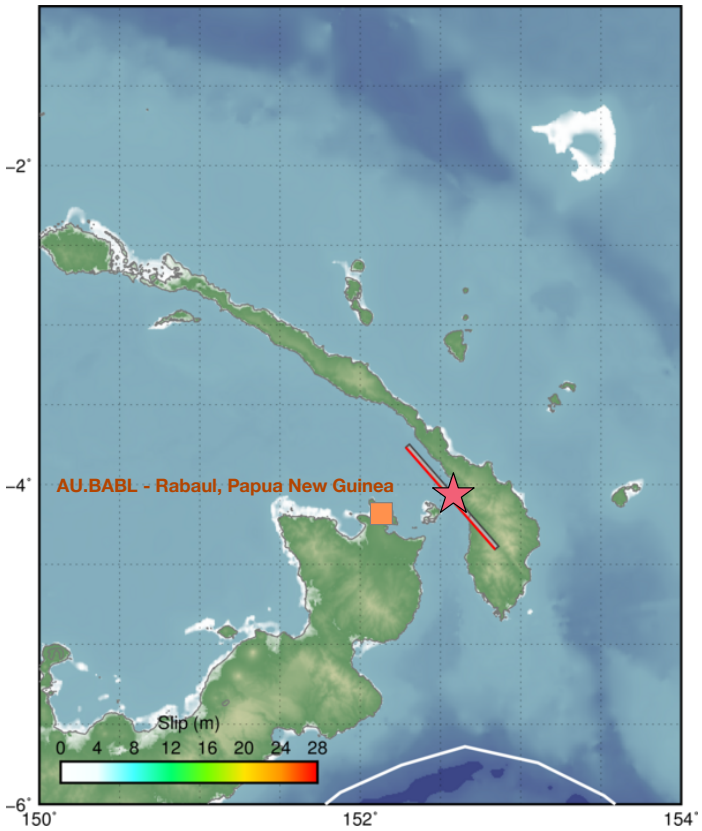
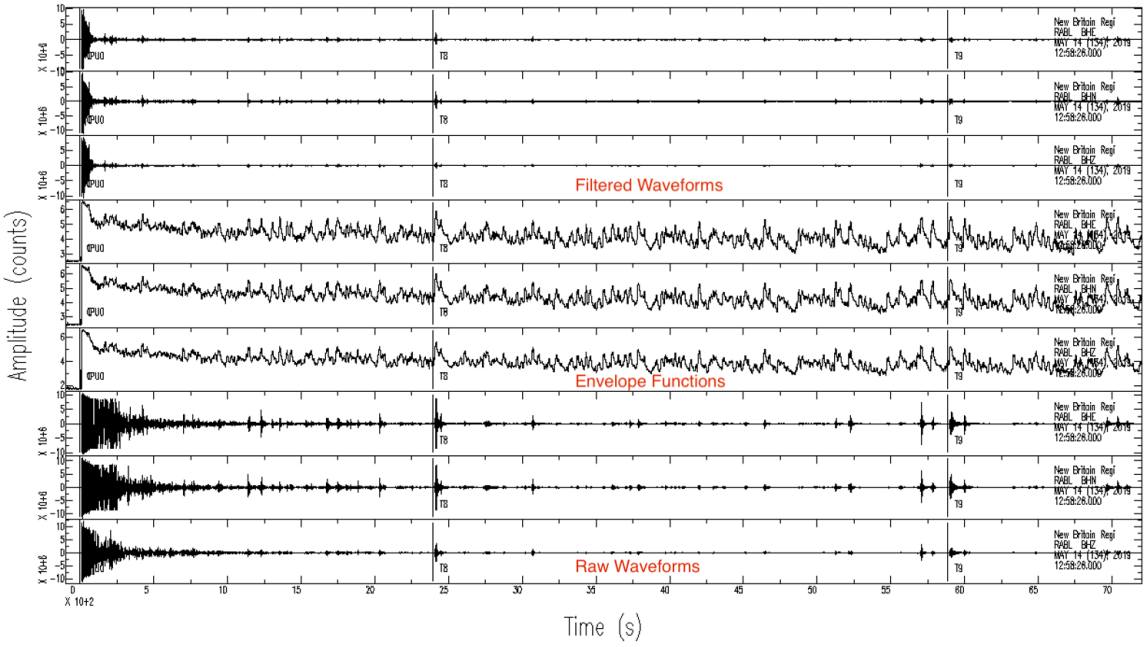
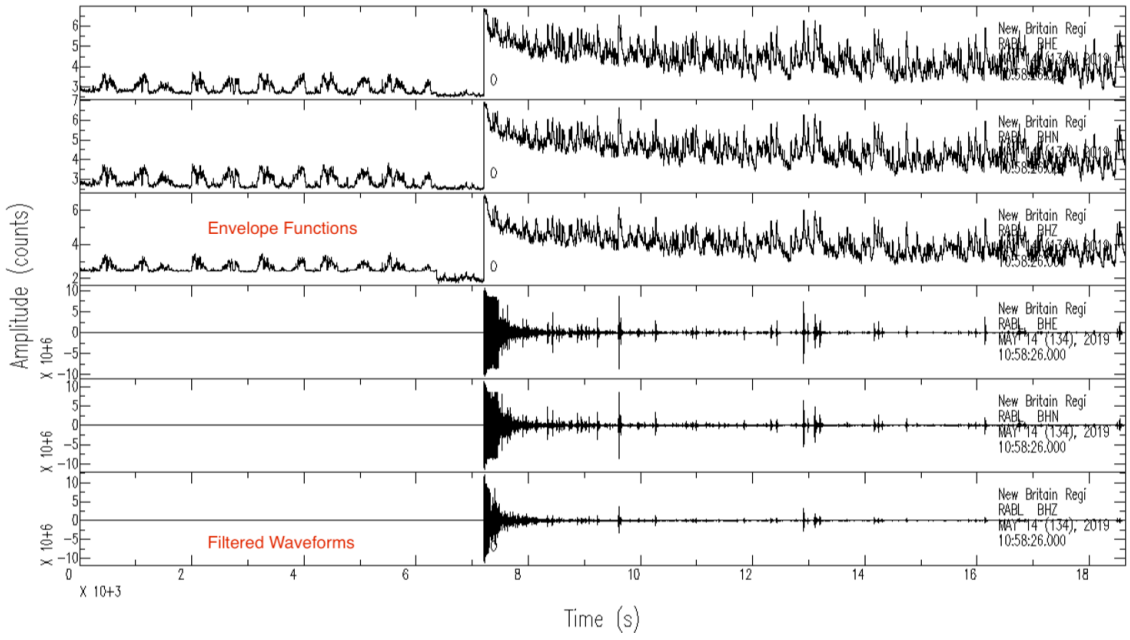


Figure 2.9: Map of study region for M_W 7.6 earthquake near Kokopo, Papua New Guinea. The main shock is marked by red start and the nearest broadband station is marked by orange square. Surface projection of the slip distribution superimposed on GEBCO bathymetry. Thick white lines indicate the major plate boundaries identified in Bird [113].

of the known unrelated information as possible (by using filtering, detrend, and spectrogram) to make the NN models focus on what is important. Zhu *et al.* [8], Zhu and Beroza [6], and Zhou *et al.* [100], however, chose to perform minimum pre-processing steps on the raw seismic waveform in order to explore the limitations on "expressiveness" of the CNN. It is believed that a sufficiently complex CNN can take the necessary data manipulation, such as band-pass filtering, into account if it is learned to be significant to the final classification task. Since no significant difference is observed in the final classification accuracy nor the complexity of the NN models between these two types of approaches, the exact choice of pre-processing can be determined by other constraints such as the availability of a dedicated DSP module on board. However, the soft-clipping method used in Zhu *et al.* [8] is particularly useful when dealing with datasets that have a broad dynamic range.



(a) Waveform of the Kokopo earthquake in the first two hours after the main shock.



(b) Waveform of the Kokopo earthquake zoomed in to the 30 minutes after the main shock.

Figure 2.10: Seismic waveforms recorded on station AU.BABL near Rabaul, Papua New Guinea after the main event of the M_W 7.6 earthquake near Kokopo, Papua New Guinea.

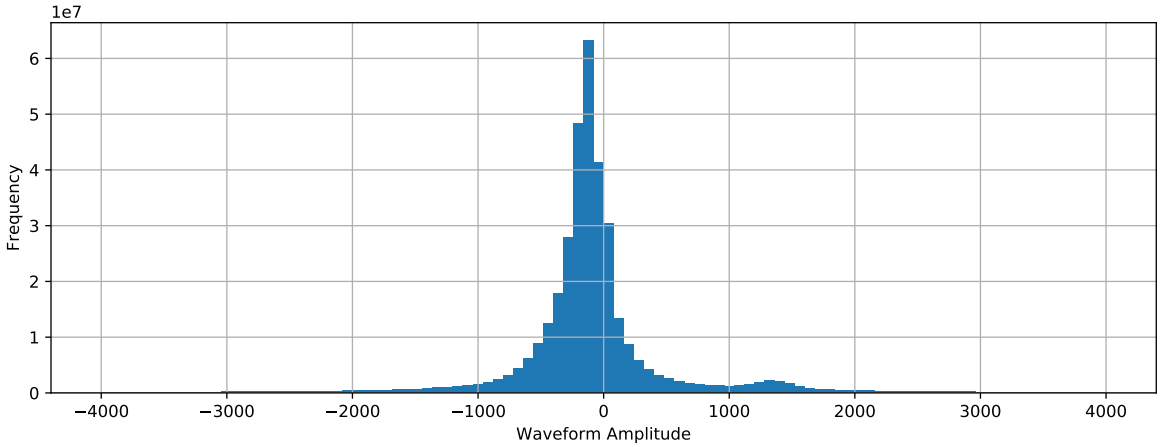
Soft-clipping Method This nonlinear preprocessing method was first designed to accommodate the large dynamic range of the waveforms in the Wenchuan dataset, where weak events are masked by stronger ones due to their amplitude difference. In the CNN, higher precision may be required after batch normalization due to such differences. Since the GPU used in this study works more efficiently for single-precision floating-point numbers, the dynamic range also imposes a hardware challenge. Hence, we apply a *soft clipping* process based on a logistic function,

$$f(x) = 1/(1 + e^{-kx}) \tag{2.3}$$

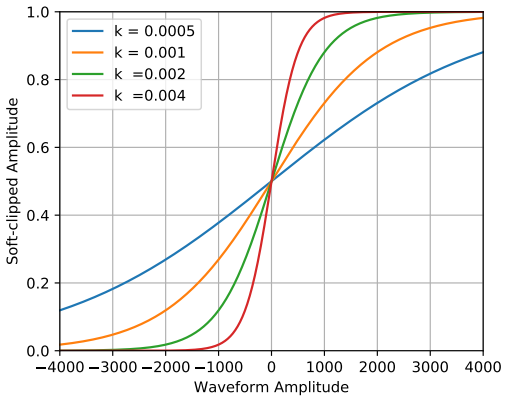
where x is the original amplitude, and k is chosen empirically based on the maximum amplitude in the original signal. The logistic functions with different k values are shown in Figure 2.11b to demonstrate varying clipping levels.

The soft clipping process, which is applied to all labeled data and continuous data with the same k value, keeps the input data range between 0 and 1, as well as reducing the relative amplitudes of strong to weak events. Figure 2.11c illustrates that soft-clipping only suppresses the large amplitude signal while keeping the small one unchanged. Figure 2.11a shows that the amplitude of most traces is less than 4000, thus we chose $k = 0.001$ and the resulting soft-clipping function is shown in Figure 2.11b.

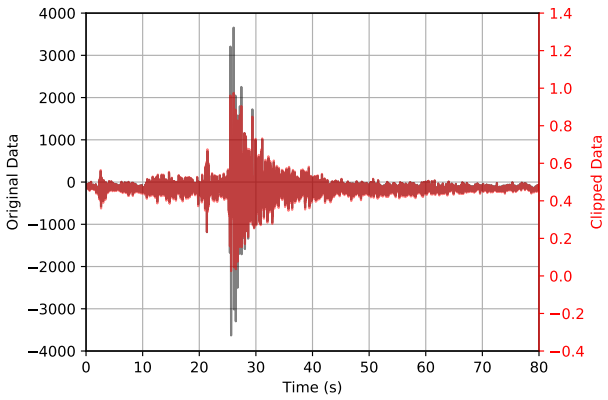
Effect of Soft-clipping During the CNN training process, the network is tested after every epoch to evaluate its accuracy. Figure 2.12 shows the training loss, defined in equation (2.2), and testing accuracy, defined in equation (2.5), versus the number of epochs. The proposed network with soft clipping (red) reaches 97% accuracy after 40 epochs and becomes stable even though the training loss keeps going down. On the other hand, without soft clipping (blue and green), the validation accuracy of the network slowly increases but exhibits a large oscillation centered around 80% and 85% accuracy, even though the training loss continues to decrease. Thus with proper preprocessing, the trained CNN can reliably determine if a given 20-s time window contains a P wave, S wave, or noise phase,



(a) Distribution of waveform amplitude.



(b) Logistic function with different k.



(c) Soft clipping effect

Figure 2.11: Pre-processing for CPIC showing (a) waveform amplitude distribution; (b) soft clipping with a logistic function on the input data and (c) example of a soft-clipped signal. Note that large amplitude signals in the original input (black) are reduced significantly on the clipped signal (red) while the small amplitude part is unchanged.

and assess the likelihood of that decision is correct.

2.4 Post-processing

Once the pre-processed seismic waveform is classified by the CNN model trained using the method described in Section 2.1.2, the output CNN scores are then post-processed for seismic event detection and phase arrival pickings. Both of these steps rely on a moving window structure to control their time resolution in detection and picking. However, once the CNN scores are computed, they use a different scheme to post-process the score. In

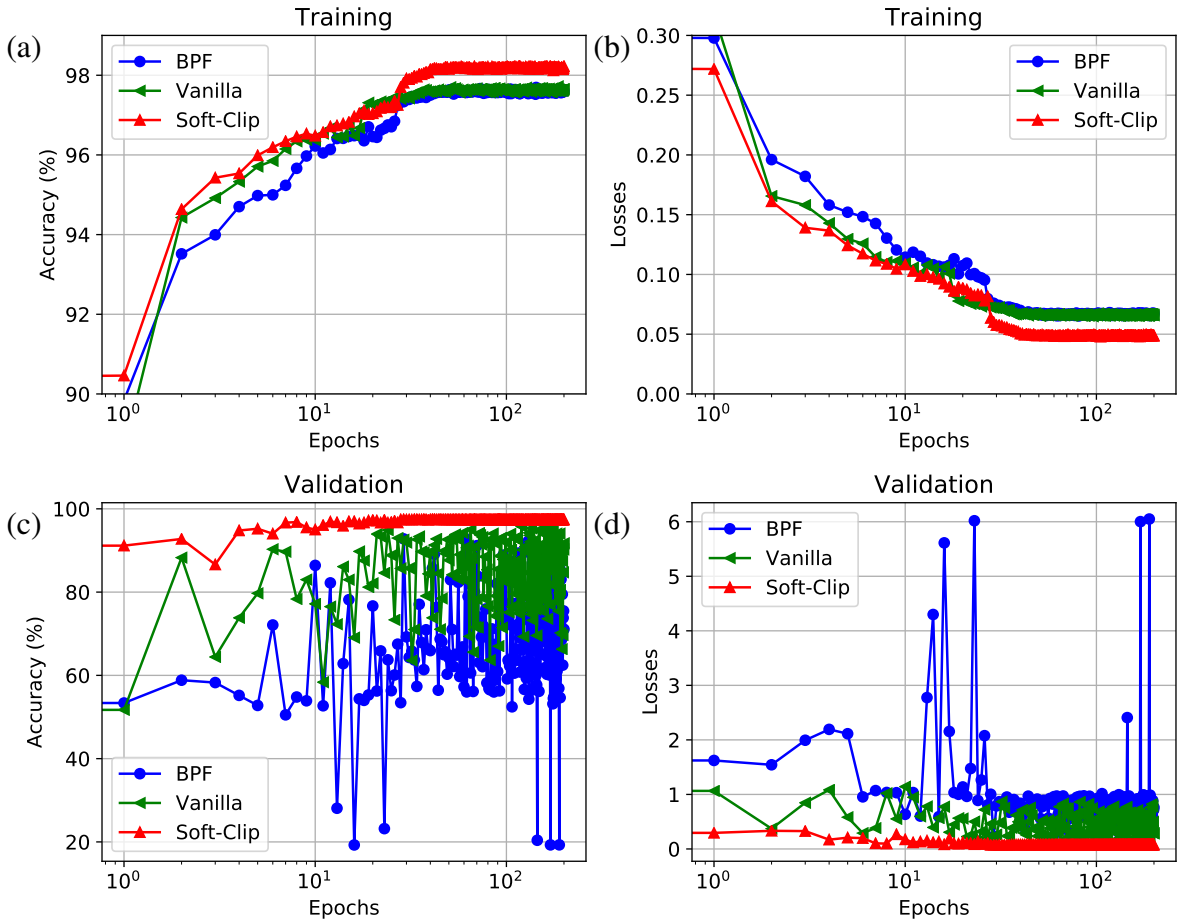


Figure 2.12: Training process of 80%–20% chronological split with different preprocessing schemes: (a) Accuracy on Training Set, (b) Loss on Training Set, (c) Accuracy on Validation Set, (d) Loss on Validation Set. Soft-clip via logistic function in (c) is the most stable method and reaches highest validation accuracy.

the event detector, the CNN scores are converted into probabilities of P-wave, S-wave, and noise classes using the *softmax* function defined in (2.1). For phase picking, the CNN scores on all time window are converted into a time-sequence which serves as the CF, analogous to STA/LTA defined by Allen [66] in Section 1.1.2. A peak detector is then applied to the generated CFs to get the arrival times. An example of post-processing is given in Figure 2.13 and the details of each step are explained in the following sections.

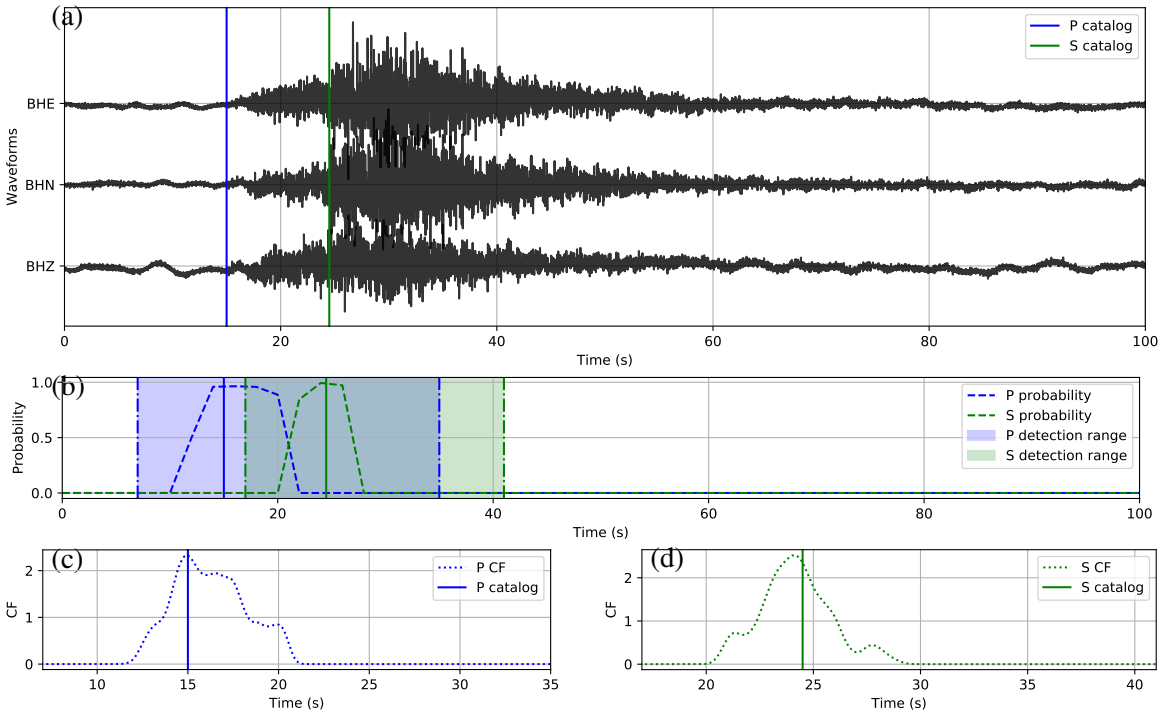


Figure 2.13: CPIC work flow: (a) Three-component waveforms (catalog P and S arrivals marked) are the input; (b) probabilities of both P and S phases calculated every 2 s from which the P and S detection ranges (shaded) are selected, starting 5 s before the first nonzero probability sample, and ending and 15 s after the last. (c, d) Arrival times picked on characteristic functions (CFs) calculated every 0.1 s within each detection range in (b).

2.4.1 Phase Detector

The phase detector in Figure 2.1 for continuous processing works on the CNN classifier outputs from moving windows that are coarsely sampled. The three outputs from the CNN classifier are converted to probabilities of noise, P phase, and S phase at each window position by (2.1). A peak probability above 0.5 is sufficient for detecting a P-phase or S-phase window. Every positive detection provides a candidate 20-s window that may contain P or S phases. Overlapping windows with the same phase label are merged into one longer window before passing to the phase picker. A detection example of a typical 100-s waveform is provided in Figure 2.13b.

The threshold 0.5 for event detection is chosen from the precision-recall trade-off curve shown in Figure 2.14 because it gives the highest precision with a recall larger than 0.95.

Notice that one can remove the constraint that a detected phase needs to have a probability higher than the noise class when weak events are sought in a low-SNR scenario. However, this practice, which increases the false alarm rate and results in a lower precision, is not recommended. This low-precision-high-recall region is not shown in Figure 2.14, but it would extend the curves further to the right. Note that the confusion matrix is shown in Table 2.2 reflects the best amount of data points for P and S phases in this plot.

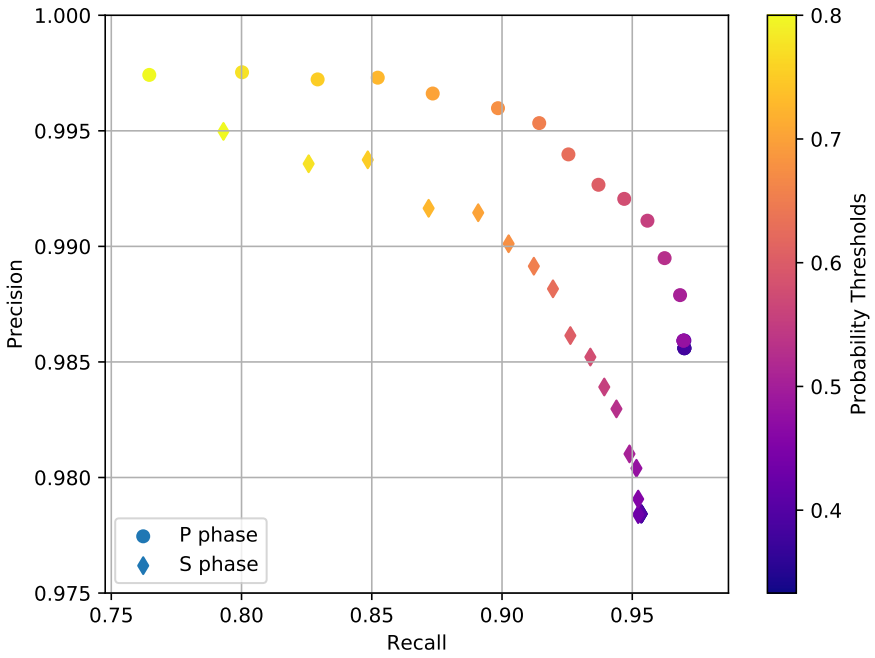


Figure 2.14: Precision-recall curve for P and S phase detection under different probability thresholds. The top left is the high-precision-low-recall region, and the bottom right is the low-precision-high-recall region. A threshold of 0.5 gives the highest precision with recall larger than 0.95. Only P or S phases with a probability higher than both the noise and the threshold are valid detections. This results in the effective minimum threshold at 0.33 for this tri-class classifier.

2.4.2 Phase Picker

The phase picker in Figure 2.1 recomputes the CNN classifier outputs over the detected windows with a smaller offset to obtain the resolution needed for accurate time picking. Since the window of P and S phases starts 5 s before the picked arrival time, the proba-

bilities output from the CNN classifier also reflect the likelihood of phase arrivals at 5 s from the beginning of the given window. Thus, the probability of each phase (arriving at 5 s within the corresponding window) should reach a local peak at the exact arrival time. Instead of using the probabilities of each phase directly, the *phase picker* relies on CFs computed as the log ratio between probabilities of each phase against the noise class as

$$\text{CF}(t) = \log \frac{p_{\text{phase}}(t)}{p_{\text{noise}}(t)} \tag{2.4}$$

Using a ratio between phase and noise probabilities makes the constructed CFs adaptive to changing noise levels. This helps to eliminate false picks caused by background noise. To achieve stable picking results, a Gaussian smoothing function with a standard deviation of three samples is applied to the CF in (2.4). Picking examples of P and S phases on the detected windows from Figure 2.13b are given in Figures 2.13c and 2.13d, respectively. Comparing to the probabilities in Figure 2.13b, CFs emphasize the arrival times of P and S phases and suppress the significance of their coda waves.

However, it is possible that multiple picks are present in one single detection window. CPIC does not force a single pick in one window; instead, it assigns a confidence level to each pick. This confidence is measured by the peaks' relative *prominence*, which is defined as the vertical distance between the peak and its lowest contour line [116]. This measure makes the picking process parameter-free; however, one can specify a minimum confidence level, e.g., $1/(n+1)$ where n is the number of picks, for a multiple-pick scenario. For example, suppose there are three picks with confidence levels as $(0.4, 0.45, 0.15)$. A 0.25 threshold of confidence would reject the pick with 0.15 prominence while keeping the first two picks. Notice that setting a 0.5 confidence threshold effectively forces a single pick in a detection window.

Table 2.2: Definition of confusion matrix for evaluating phase detector

		Detector			
		Noise	P-wave	S-wave	Total
Catalog	Noise	N_n	N_p	N_s	$N_n + N_p + N_s$
	P-wave	P_n	P_p	P_s	$P_n + P_p + P_s$
	S-wave	S_n	S_p	S_s	$S_n + S_p + S_s$
	Total	$N_n + P_n + S_n$	$N_p + P_p + S_p$	$N_s + P_s + S_s$	ALL

2.5 Model Performance

2.5.1 Performance Evaluation

CNN Classifier We can evaluate a CNN classifier by processing labeled testing data where the true output is known. The *accuracy* defined below is a simple measure of a classifier's performance:

$$\text{accuracy} = \frac{\text{number of correctly labeled samples}}{\text{number of all testing samples}} \tag{2.5}$$

Noise labels are not treated differently from phase labels, so classifying a noise window correctly has the same weight as confirming a phase window.

Phase Detector The detector can be viewed as a three-class classifier that decides whether a given time window contains a seismic phase (P or S), or only noise. To evaluate the detector's effectiveness, we use a confusion matrix as in Table 2.2, where the labeled windows of each class (per row) are sorted into the number of each detected type (per column). Subscripts denote the detected class, e.g., P_s is the number of windows with P-phase labels but detected as S-phase. The sum of all nine counts equals the total number of labeled windows in the given catalog. To avoid the effect of an imbalanced dataset dominated by noise windows (large N_n), we can use $precision$ and $recall$ (a.k.a. sensitivity) for each class to

measure the performance, which ignores N_n. These are defined for the P-wave class as:

$$\begin{aligned} precision: \quad \mathcal{P}_p &= \frac{P_p}{N_p + P_p + S_p} \\ recall: \quad \mathcal{R}_p &= \frac{P_p}{P_n + P_p + P_s} \end{aligned} \tag{2.6}$$

$\mathcal{P}_n, \mathcal{P}_s, \mathcal{R}_n$ and $\mathcal{R}_s$ can be defined similarly. Notice that both $precision$ and $recall$ are independent of N_n. Ideally, both $\mathcal{P}$ and $\mathcal{R}$ for each class would be close to 1, However, the labeled aftershock dataset catalog we have is incomplete – it tends to include only the strong and obvious phases while omitting weak events. Thus, higher N_p and N_s counts are expected which lowers $\mathcal{P}_p$ and $\mathcal{P}_s$, although some of these N_p and N_s detections are likely weak phases not listed in the catalog. On the other hand, $\mathcal{R}_p$ and $\mathcal{R}_s$ should be high if very few manually labeled strong phases are missed. Notice that the accuracy defined in (2.5) measures the ratio between the sum of diagonal terms over all terms in the confusion matrix:

$$accuracy = \frac{N_n + P_p + S_s}{ALL}$$

Similarly, to avoid a dominant N_n count biasing the accuracy, often the F-1 score is computed from $precision$ and $recall$ (their harmonic mean) for each class:

$$\text{F-1} = \left(\frac{precision^{-1} + recall^{-1}}{2} \right)^{-1} \tag{2.7}$$

Phase Picker The phase picking process estimates the arrival time for each detected seismic phase. We measure our phase picker's error as

$$E_{\text{pick}} = T_{\text{pick}} - T_{\text{cat}} \tag{2.8}$$

where T_{pick} is the arrival time from CPIC and T_{cat} is the manually picked phase arrival time. Then the systematic bias and variance of our phase picker estimator are measured by taking the mean and standard deviation of E_{pick} over all phases in the catalog. We expect a close-

Table 2.3: Confusion matrix for phase classification on the validation dataset which is the latest 20% of the labeled phases.

		Detector			
		Noise	P-wave	S-wave	Total
Catalog	Noise	$5,946$	97	113	$6,156$
	P-wave	22	$2,930$	10	$2,962$
	S-wave	59	6	$2,873$	$2,938$
	Total	$6,027$	$3,033$	$2,996$	$12,056$

Table 2.4: Precision, recall, and F-1 score for the three classification categories.

Categories	Precision	Recall	F-1 Score
Noise	0.987	0.966	0.976
P-wave	0.966	0.989	0.9787
S-wave	0.959	0.978	0.968

to-zero bias and reasonably low variance even though the catalog picks may contain some human error. Note that catalog phase arrival times are rounded to the tenth decimal point (0.1).

2.5.2 Training and Validation

To systematically verify the accuracy and stability of the proposed CNNs, the available 60,000 labeled windows are split into a training subset and a testing subset. The split is done chronologically to emulate a real-world scenario: training on historical phases (80%) and testing on future ones (20%). The training process involves minimization of the loss function (2.2) with iterative updating based on the gradient. After the CNN training process sees every sample in the entire training dataset once, we have finished one *epoch* of training. At the end of each epoch, we generate a testing result to score the CNN classifier accuracy and thus track the progress of its training. Multiple epochs are needed to fully train the CNN weights into a stable state.

Reliable Classifier As demonstrated in Figure 2.15, the training process of the proposed CNN converges after 40 epochs; no over-fitting is observed even after 200 epochs. The

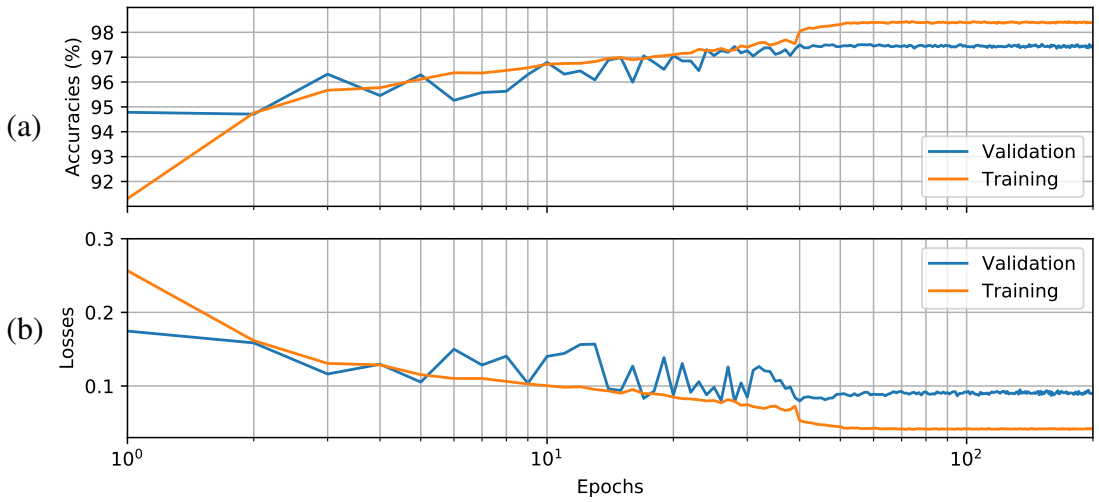


Figure 2.15: Training performance: (a) classifier accuracy and (b) loss function against number of epochs on training and validation datasets during the CNN training process.

overall validation accuracy of this experiment reaches 97.5%, using the diagonal entries of detailed confusion matrix shown in Table 2.3. Precision, recall, and F-1 scores are given in Table 2.4. To further understand the characteristics of the trained CNN, we grouped the testing dataset into smaller bins sorted by event magnitude, source-receiver distance, and SNR. The trained CNN is validated on these small testing datasets, and their F-1 scores are plotted in Figure 2.16. The results generally follow our intuition: phases associated with events of larger magnitudes (Figure 2.16a) and smaller distances (Figure 2.16b) being classified with higher accuracy. Figure 2.16c demonstrates that the F-1 score is inversely proportional to the waveform SNR for both P and S phases.

Flexible Training Set Size As mentioned before, the overall 60,276 samples are split into training and validation datasets chronologically with different splitting ratios to explore the minimum required training dataset size. Each split is trained up to 200 epochs and the model accuracy defined in (2.5) is shown in Figure 2.17. In general, the relationship between training set size and validation accuracy follows a log function as demonstrated in Figure 2.17. We note that CPIC reaches 95% accuracy with less than 6,000 training samples and 97% with less than 30,000 training samples. This largely reduces the amount

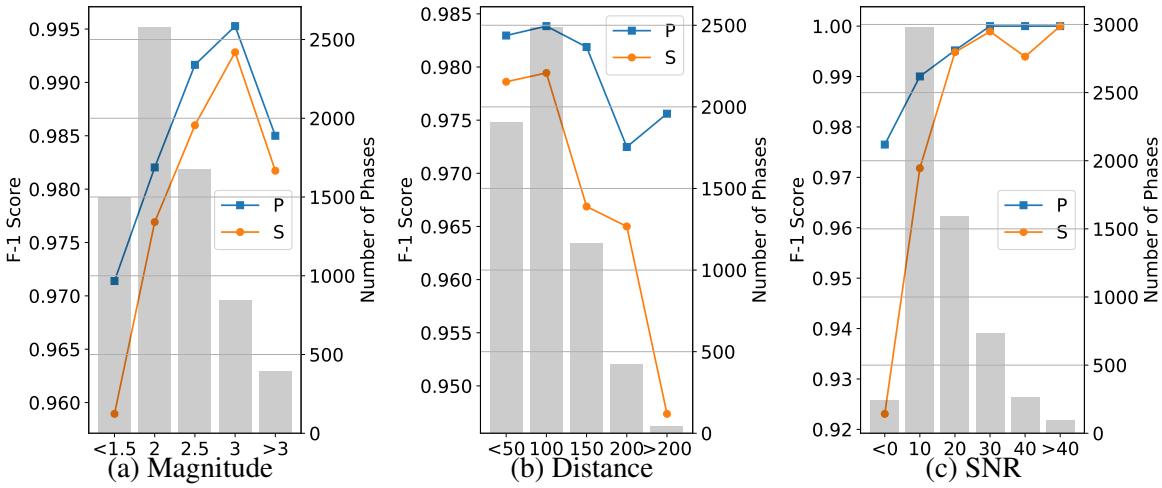


Figure 2.16: F1 scores (right axes) of the trained classifier versus (a) magnitude, (b) distance, and (c) SNR. P (blue) and S (orange) phases are plotted separately. The number of testing samples in each small bin (left axes) is shown by the bars in the background.

of manual labeling needed to a reasonable level for practical applications. For example, CPIC would only require 300 manually picked aftershock events (for both P and S phases) per station on a 10-station network to achieve 95% classification accuracy.

Fast Deployment CPIC is tested using the Nvidia GTX 1080 Ti GPU with 3,584 CUDA cores and 11 GB memory. The PyTorch machine learning package [117] and glsobspy seismic processing toolbox [118] were used to automate the testing. Online processing of one 20-s window by the trained CNN takes less than 0.3 ms on average when feeding the input as 1000 windows per batch to exploit the maximum GPU memory size. This enables us to run the detector on the entire 31-day continuous 3-C waveforms recorded by 14 stations within two hours. The time spent for phase picking depends on the number of detected phases and the merged window length. In our study, it takes around 12 hours to pick all 30,000 catalog phases within the 31-day dataset.

2.5.3 Detection on Continuous Waveforms

With a 2-s offset, the continuous waveforms are broken into a collection of 20-s overlapped time windows for detection (see Figure 2.1). CPIC gives a label to each such 20-s window

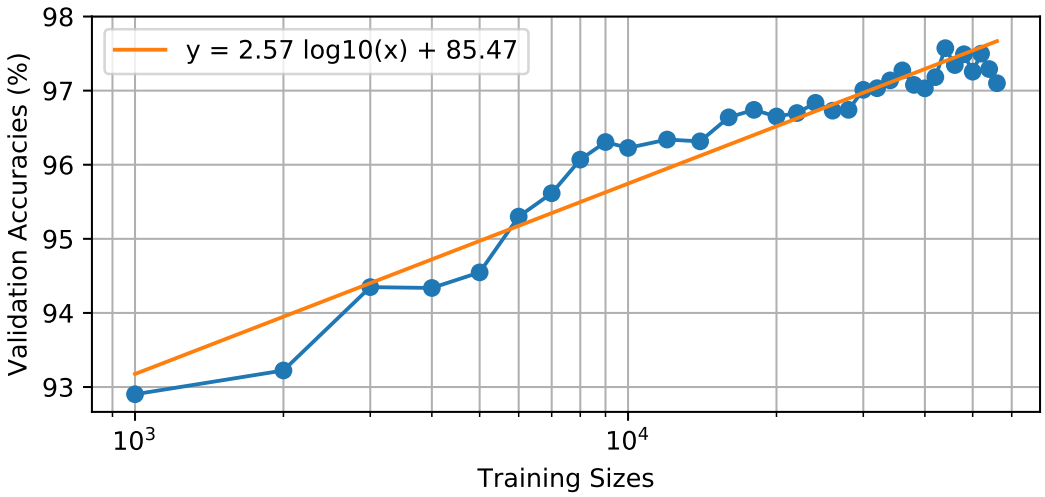


Figure 2.17: Validation accuracies vs. training dataset sizes (log scale) in blue. A line (log function) is fitted in orange.

as P phase, S phase, or noise. Consecutive windows with the same label are merged into one longer window (Figure 2.13b), e.g., four neighboring 20-s windows expand to a 28-sec window. As shown in Table 2.4, 98.6% and 97.8% of the catalog P and S phases are correctly detected (recall), while 97.0% and 95.4% detected P and S phases match a catalog phase (precision).

Figure 2.18 shows the application of the CPIC detector on a 15-minute continuous section across all 14 stations. For the three catalog events (M_L 1.6, 2.6, and 2.1, respectively), the CPIC detector finds all phases picked in the catalog (marked by vertical bars in red for P phase and magenta for S phase). Moreover, it detects additional phases for these three events on other stations that were missed by manual picking, e.g., P (blue peak) and S (green peak) phases around 400 s on five additional stations (SPA, QCH, PWU, MIAX, and WXT) for the M_L 2.6 event.

On the other hand, additional phases are also detected, which might be associated with events missed in the catalog. For example, two clusters of phases around 80 s and 300 s in Figure 2.18 exhibit reasonable moveout curves and may correspond to legitimate events. To investigate these additional phase detections, we built a matched-filter (MF) enhanced catalog for one day (8/30/2008) following the procedure used by [119]. The analysis pro-

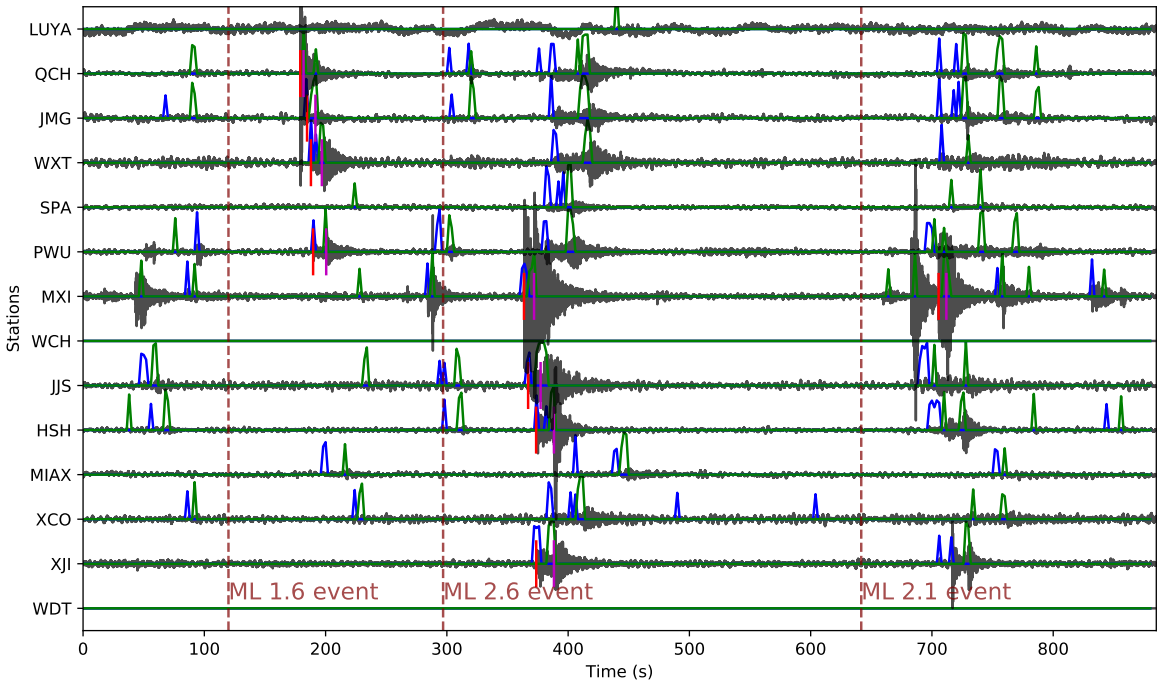


Figure 2.18: Detection example on 15-minute recording on 14 stations with three catalog events. Only vertical components are plotted. Blue and green curves show the probabilities of P and S phases. Red and magenta bars indicate the catalog P and S arrivals. Origin times of three catalog events are marked by the dashed vertical lines along with their magnitudes.

cedure of matched filter detection generally follows [119] and is briefly described here. Over 6,500 cataloged events between 2008/08/01 and 2008/08/30 are used to extract 6-s templates. A 2–8 Hz band-pass filter is applied to enhance the strength of local earthquake signals, and the filtered waveforms are downsampled to 20 Hz. The 6-s template window starts 1 s before either the P wave on the vertical component or the S wave on horizontal components. To avoid noisy traces, we measure the noise energy in a 6-s window ahead of the template and define the corresponding SNR as the ratio between the energy of the template and the noise energy. Only traces with SNR above 5.0 are used to cross-correlate with continuous data and output the cross-correlation function. Stacked cross-correlation values on multiple stations are used to detect candidate events with a threshold of nine times the median absolute deviation (MAD) of the daily stacked correlation trace. We select 2008/08/30 as the testing day since it has the most cataloged events, approximately 300. Eventually, we end up with approximately 1,300 events and 12,200 phase picks which

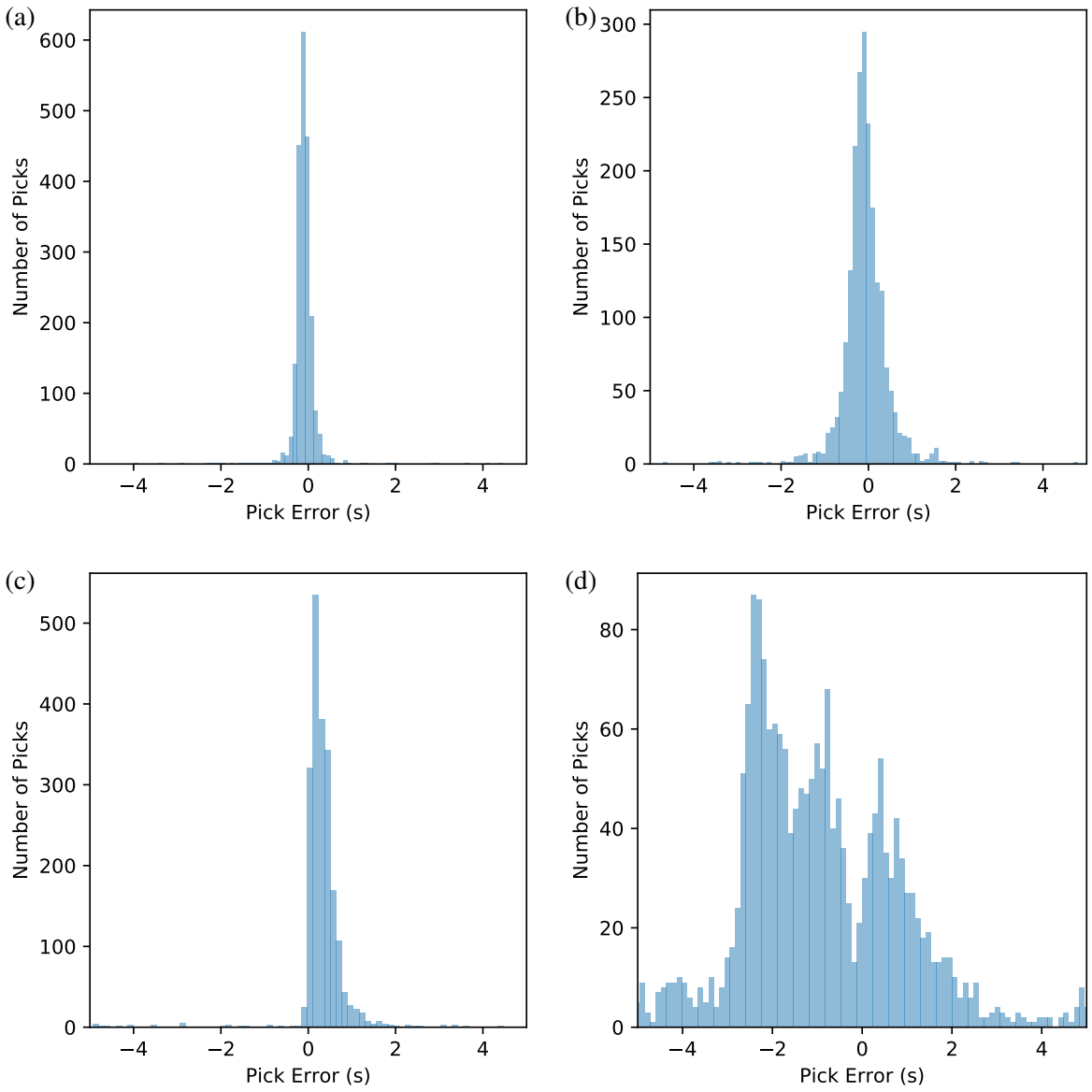


Figure 2.19: The distributions of picking errors (E_{pick}) of CPIC (upper panels) and ObsPy AR picker (lower panels) on the validation dataset.

are detected on at least three stations.

During the same time, CPIC detects 4,123 seismic phases among which 2,892 (70%) contain a phase in the MF catalog. Further studies are needed to check whether the remaining 30% correspond to actual events that are not similar to existing templates.

Table 2.5: Evaluation metrics for CPIC and ObsPy AR picker on the validation dataset.

Method	$\mu(E_p)$	$\mu(E_s)$	$\sigma(E_p)$	$\sigma(E_s)$
CPIC picker (ms)	-79.0	-78.9	138.8	293.0
ObsPy AR picker (ms)	311.4	936.3	671.6	1,697.0

2.5.4 Picking for Catalog Arrival Times

Picking Results The detected windows are reprocessed by the CNN with a 0.1 s offset to generate the CPIC arrival times. The picked arrival times are compared with the catalog phase arrivals and results from the ObsPy AR picker. The error defined in (2.8) is used to measure the performance of the P and S phase pickers separately. Table 2.5 summarizes the statistics of picking errors for P and S phases from CPIC and the glsobspy glsar picker. Errors for both P and S phases from CPIC have much smaller standard deviations and biases than their counterparts from the glsobspy glsar picker. Significant improvements are observed by applying CPIC, especially for S-wave arrival times. This is expected since picking S phase arrivals is more challenging for traditional methods due to interference from the P wave coda. Figure 2.19 compares the distributions of picking errors for P and S phases from CPIC with the glsobspy glsar picker. The error distributions from both methods for P arrivals are narrower than those for S waves. This is consistent with our intuition that P phase arrivals are clear and easier to pick. Notice that both distributions from CPIC are more symmetric than those from the ObsPy AR picker.

Picking Examples Examples of arrival picking are given in Figures 2.20 and 2.21 to demonstrate CPIC's performance. Note that the waveforms displayed in the upper panels have their mean removed and are scaled to have a maximum amplitude of one; however, the real inputs to the CPIC model are the original raw waveforms. Figures 2.20a and 2.20b show ideal cases where there is only one distinct peak in the CFs of both P and S phases that aligns perfectly with the catalog arrival times. Multiple peaks are present in Figures 2.20c and 2.20d, but the picks made by CPIC correctly match the manual picks. Less ideal cases

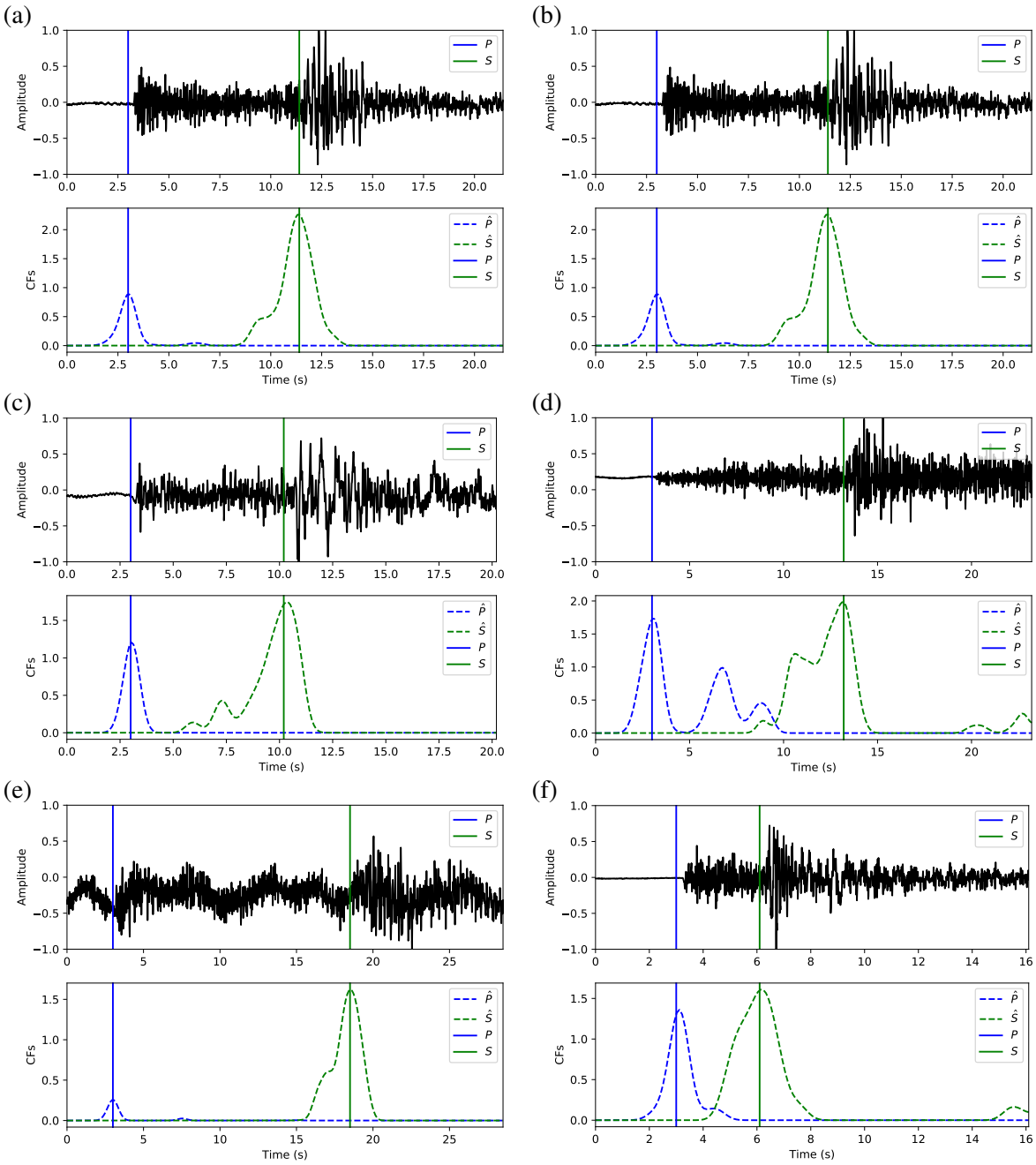


Figure 2.20: Examples of CPIC picks that are consistent with manual picks. The upper panels of (a) – (f) are the vertical components from the 3-C waveforms used in the picking process. Vertical lines denote arrival-time picks. The lower panels show the characteristic functions (CFs) of $\hat{P}$ (blue) and $\hat{S}$ (green) used by CPIC to pick the arrival times.

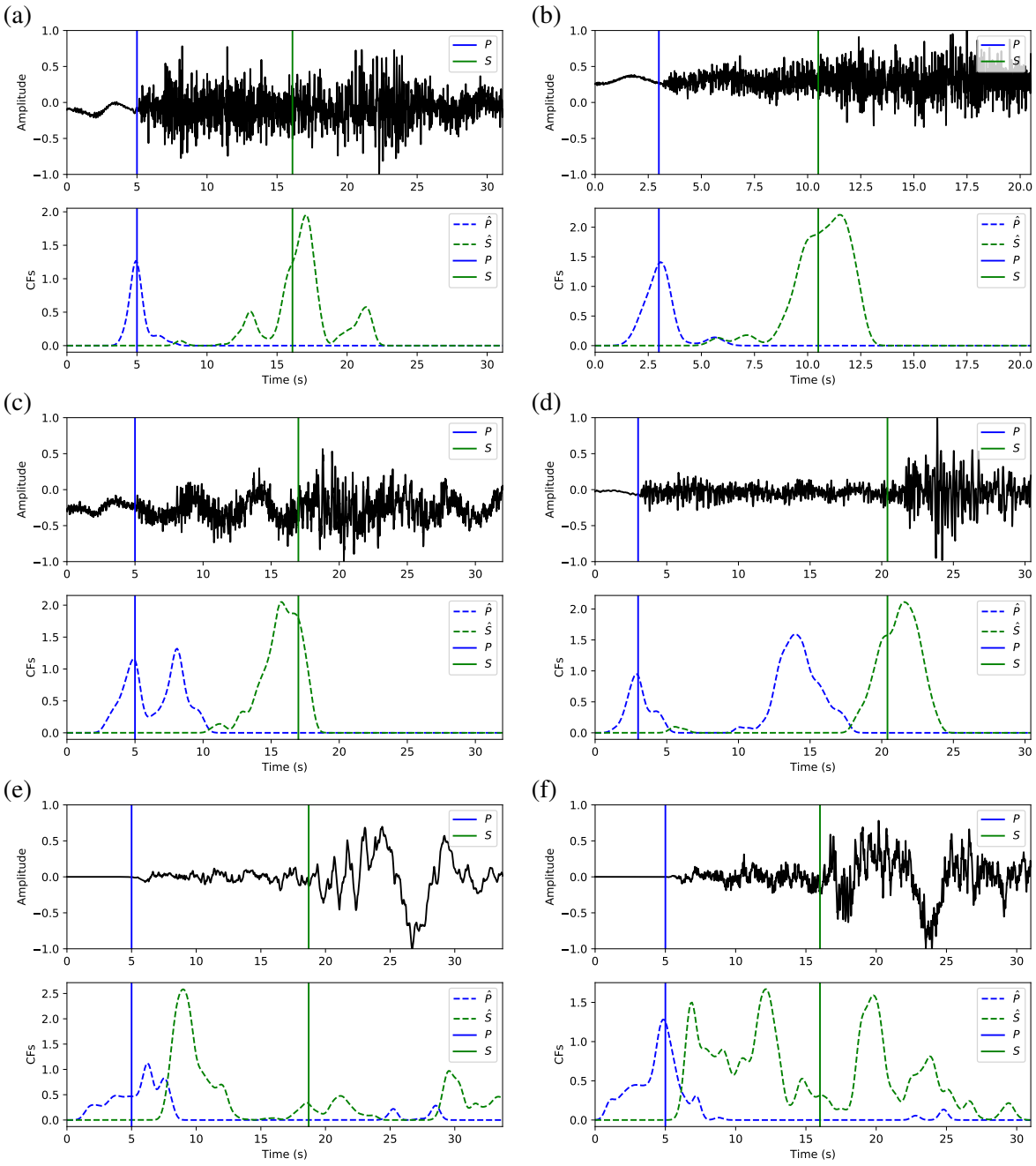


Figure 2.21: Examples of CPIC picks that are inconsistent with manual picks. (a, b) are examples of ambiguous S picks. (c, d) are examples of multiple P picks. (e, f) are examples of an M_L 6.1 events on two distant stations.

are shown in Figures 2.20e and 2.20f where CPIC picks the correct arrival times, but may have issues when the conditions are worse. The noisy waveform in Figure 2.20e results in a small peak for a P wave around 3 s, which may be buried under the noise floor if more severe noise were present. CPIC picked the arrival times in Figure 2.20f correctly, but has a small tail for the S phase at the end. This small tail was successfully rejected due to its small amplitude, but it may become a false alarm if the relative peak amplitude of the S phase around 6 s were much smaller. This is also the case for Figure 2.20d. Examples of picks inconsistent with the catalog arrival times are also shown in Figure 2.21. Unlike the multiple peak cases shown in Figure 2.20, the peak CFs from CPIC in Figures 2.21c and 2.21d are more than 1 s from the manually picked arrivals. Figures 2.21e and 2.21f show incorrect picks of a M_W 6.1 event on two distant stations (SPA and WXT). Since there are only two events with magnitude larger than M_W 6 in the given Wenchuan catalog, the trained model appears to be "inexperienced" with such large events. This is one of the disadvantages for training-based approaches: the model needs to see enough examples before it can provide reliable predictions.

2.6 Generalization to Different Datasets

2.6.1 Training on Large SCSN Dataset

The CPIC model was also trained on the SCSN dataset, which was downloaded from `http://scedc.caltech.edu/research-tools/deeplearning.html`. Note that the SCSN dataset was pre-processed as described in Ross *et al.* [4], which is different from the CPIC approach described in Section 2.3. The most notable differences are that Ross *et al.* [4] normalize the 3-C waveforms by the absolute maximum amplitude and apply a 2-Hz high-pass filter. This practice eliminates the amplitude differences between small and large magnitude earthquakes, which are known to have different frequency content.

As shown in Figure 2.22, the training process converges after 40 epochs and the testing accuracy holds steady after 200 epochs without any drop in accuracy. This indicates that

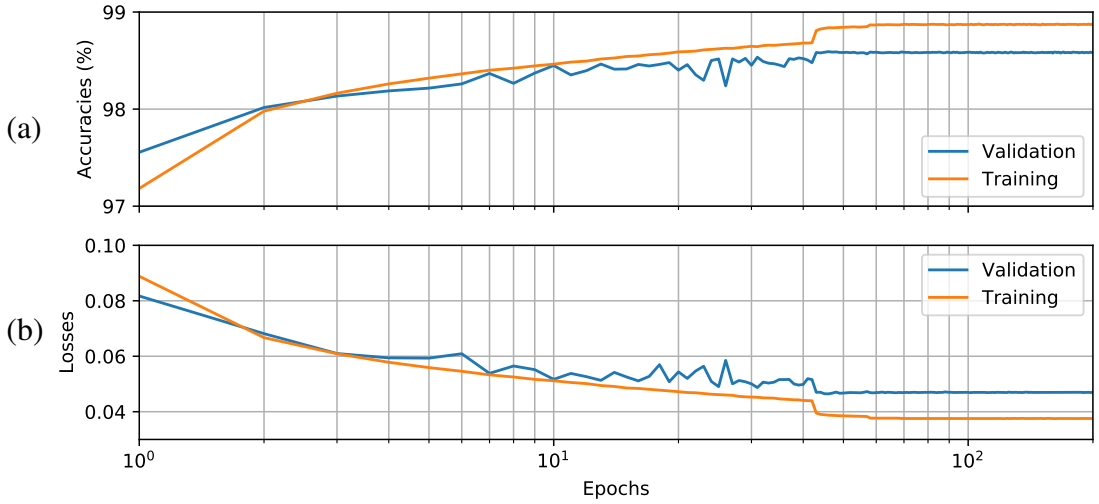


Figure 2.22: Training performance on SCSN dataset: (a) classifier accuracy and (b) loss function against number of epochs on training and validation datasets during the CNN training process.

there is little over-fitting effect in the CPIC training process. When we compare to a similar plot provided in Figure S1 of Ross *et al.* [4], where the validation loss diverges after the second epoch, the CPIC approach provides a more reliable and stable model. This is significant considering the small model and training dataset we had from Wenchuan, which might be suspected to have over-fitting or divergent validation accuracy/loss after longer (more epochs) training. This characteristic of well-behaved training is truly the strength of the CPIC approach since there is little concern that CNN model training fails to converge on a small training set. It is possible that the CPIC model might achieve higher accuracy on a much larger dataset; however, the current performance at 97.5% accuracy on the Wenchuan dataset and 98.6% for the SCSN dataset is sufficient for the targeted applications.

2.6.2 Training on Small Mariana Dataset

As described in Section 2.2.2, CPIC is trained on 80% of the 18,354 labeled samples and validated on the remaining 20%. This is a considerably smaller dataset compared to the Wenchuan and SCSN datasets. Using the same pre-processing steps described in Section 2.3, the CPIC model is trained and validated the same way as for Wenchuan and SCSN.

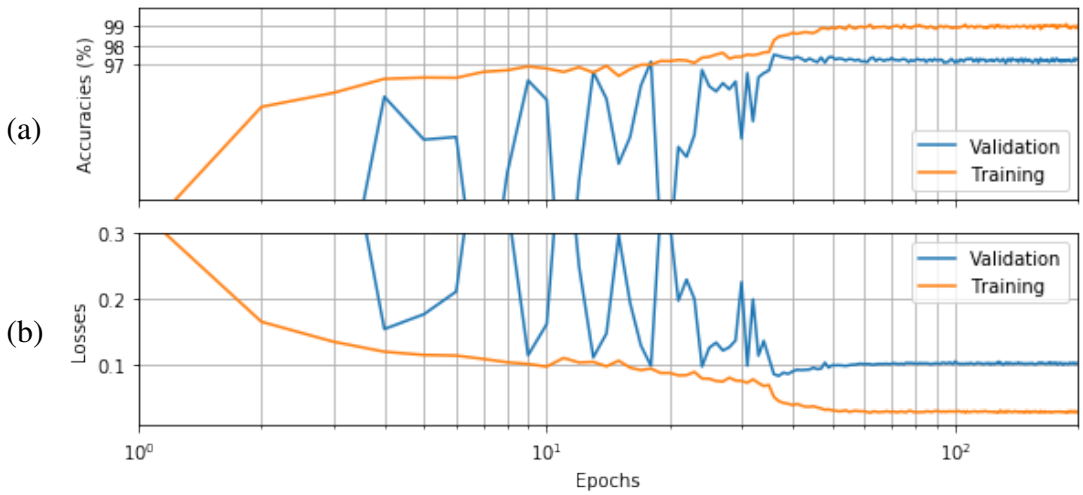


Figure 2.23: Training performance on Mariana dataset: (a) classifier accuracy and (b) loss function against number of epochs on training and validation datasets during the CNN training process.

Shown in Figure 2.23, the training process starts to converge after 50 epochs and becomes stable after 200 epochs. However, compared to the similar training plots in Figure 2.15 and Figure 2.22, the validation accuracy starts to drop slightly while the training accuracy continues to increase after 35 epochs. The same goes for the training and validation losses as well. This is an indication that the CPIC model is too complicated for the training set; in other words, more training samples are needed to train the CPIC model with stable results. Figure 2.24 demonstrates the picking results on the continuous waveforms of the Mariana dataset. The CPIC picker correctly picked most of the catalog arrivals as well as additional small events that were missing from the catalog.

2.6.3 Transfer Learning in Induced Seismicities in Okalahoma, U.S.

When we applied the original CPIC classifier trained on the Wenchuan dataset, it achieved accuracy above 90% on the two near stations (OK025 and OK029), but not on the far station (OK030) as shown in Table 2.6. Next, we retrained the model by fine-tuning only the fully-connected (FC) layer that classifies feature vectors into probabilities of phase/noise classes. After fine-tuning the classifier on approximately 2,000 samples ($\approx$ 350 events),

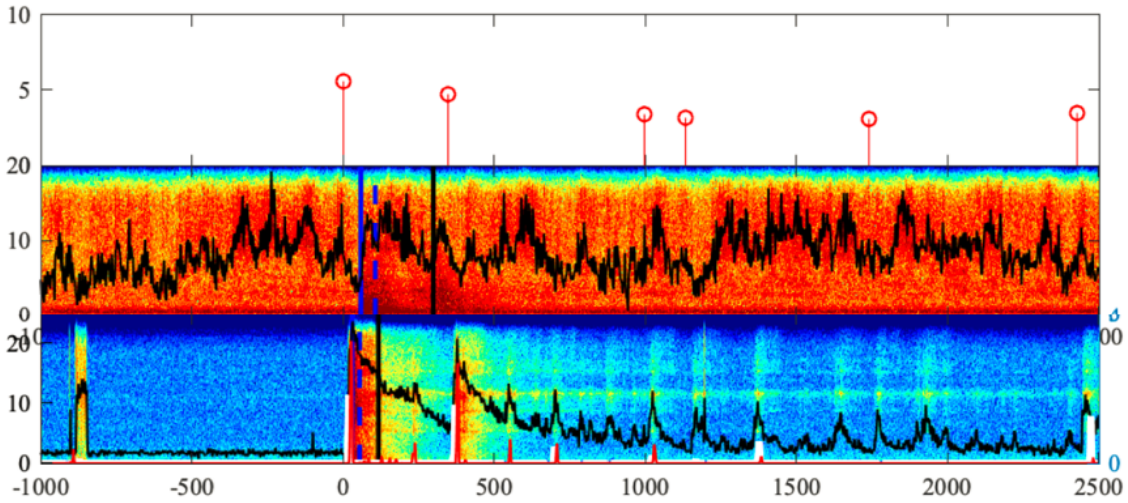


Figure 2.24: Phase picking applying trained CPIC to continuous waveform on Mariana dataset. The plots from top row to bottom row are (a) catalog phase arrivals; (b) envelope functions; and (c) CPIC picks (P by white vertical stripes and S-wave by red vertical stripes) superimposed on the spectrogram.

Table 2.6: CPIC accuracy when testing on a three-station seismic dataset in OK, USA. The first row shows the performance of directly applying CPIC as trained on the Wenchuan, China dataset, while the second row shows the enhanced accuracy after fine-tuning CPIC on 2,000 training samples from the Oklahoma region.

Station	OK025	OK029	OK030	All
Original (%)	95.7	92.2	69.9	87.5
Fine-tuned (%)	98.8	96.2	94.2	97.0

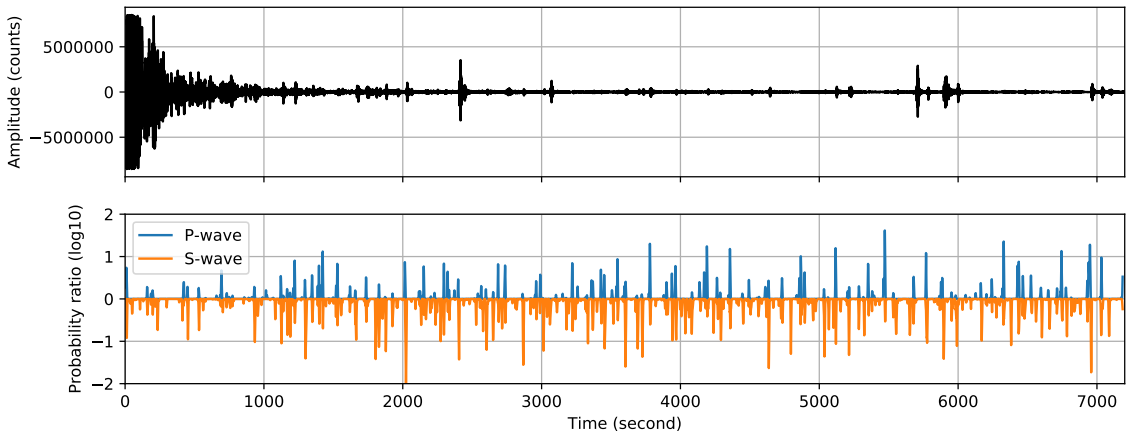


Figure 2.25: Picking results on the first two hours of the 2019 M_W 7.6 earthquake near Kokopo, Papua New Guinea: (a) two-hour continuous waveform of the vertical channel; (b) CF from CPIC model in log-probability with P-waves marked in blue and S-waves marked in orange. Note that CF for S-wave is inverted to negative values.

the accuracy on all three stations is above 94% with an overall accuracy at 97.0%. This shows that the convolutional layers in the CPIC model capture the essential representation of a seismic wave needed for phase classification. After fine-tuning the classification layer (FC), the CPIC model trained on one region can be generalized to other regions for different event types (aftershocks vs. induced earthquakes).

2.6.4 Direct Application to Aftershocks of Southern Pacific Earthquake

The CPIC model trained on the Wenchuan dataset was directly applied on the continuous waveform captured for the 2019 M_W 7.6 earthquake near Kokopo, Papua New Guinea. As described in Section 2.2.3, the two-hour-long continuous waveforms from the station AU.BABL with three components is fed into the pretrained CNN model. Using the phase picking method described in Section 2.4 as a post-processing step, both P-waves and S-waves are picked as shown in Figure 2.25. Significantly more aftershock events are identified compared to only two in the original catalog. Notably, the entire picking process was finished within one minute for the entire two-hour-long waveform. When zoomed in to verify the individual picks, several difference cases were encounter as shown in Figure 2.26.

The first panel in Figure 2.26 show the first 100 s after the main shock. Both P-wave and S-wave of the main event were correctly identified. Several weak S-waves were also noticeable; however, they are overwhelmed by the large energy from the main shock. The second panel demonstrates a special case that P-waves and S-waves are superpositioned with large low-frequency oscillations possibly due to the surface wave for the main event. CPIC is still able to identify both the P-wave and S-wave phases under strong background oscillation noises. The third panel shows a typical use case of CPIC where both P-wave and S-wave can be well defined and seen by raw eyes. Notice that both the strong event at the beginning and the weaker event at the end were correctly identified. Last but not least, the fourth panel shows a case when the soft-clipping coefficient in (2.3) was incorrectly set. In this case, the parameter is set larger than it needs to be which makes the CPIC model responding to weaker events while ignoring strong event since they are being severely clipped. The one P-wave and three S-wave phases identified in the last panel all correspond to the very weak events that are almost overshadowed by their strong neighbors. This phenomenon, though being unintentional, can be used towards our advantage. Given the same model trained for a normal event, one can tweak the soft-clipping coefficient to deliberately making the CPIC model more sensitive to stronger or weaker events without a need of retraining the CNN model.

2.7 Other Network Networks for Seismic Processing

Both CNN [4, 6, 8] and RNN [120, 100] can be applied for seismic applications. The exact choice of the neural network structure depends on the nature of the characteristics of the end task. Some studies treat them as two separate tasks and develop independent NN models for each task [4, 6, 120, 100]. Ross *et al.* [4] built a classifier for P-wave, S-wave, and noise for 4-s windowed seismic waveforms. The windows with a predicted class other than the noise are regarded as a detection. Zhu and Beroza [6] took the detected windows and trained an estimator of arrival times. Unlike Zhou *et al.* [100], which intuitively predicts

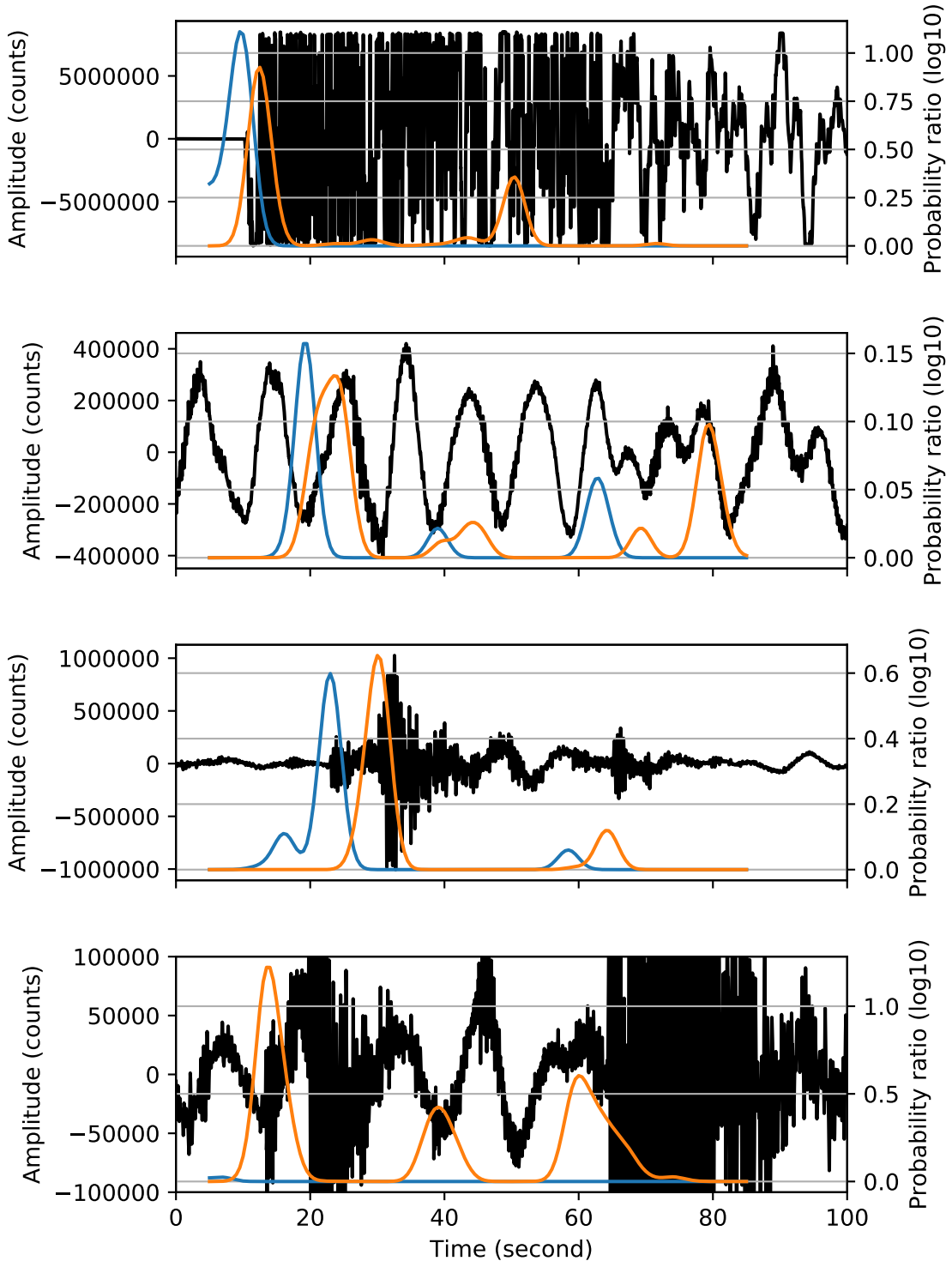


Figure 2.26: Four 100-s windows picked by CPIC with P-wave marked in blue and S-wave marked in orange on 2019 M_W 7.6 earthquake near Kokopo, Papua New Guinea: (a) right after main shock; (b) 1,000 s after main shock; (c) 2,000 s after main shock; and (d) 3,000 s after main shock with larger soft-clipping factor.

the arrival time, Zhu and Beroza [6] predicts a distribution of the arrival time over the given waveform period. PhaseNet proposed by Zhu and Beroza [6] was trained on 30-s windows given the artificial Gaussian distribution of arrival times with 0.1 s standard deviation as labels. The same group followed up with Mousavi *et al.* [120] that proposed to used a combination of CNN and RNN for accurate event detection. The usage of RNN for phase picking application is initiated by Zhou *et al.* [100], where a CNN model was proposed for event detection and a RNN model for phase picking. Zhu *et al.* [8], however, achieves both event detection and phase picking using a single CNN model. It utilizes the CF concepts from Allen [66] and use the trained CNN model on a series of overlapped moving windows as a generator of the CFs. Notice that many of these studies use a particular type of NN named CNN to model their classifiers. As mentioned in Section 1.1.1, CNN took advantage of the spatial correlation in images and use convolution operator to share weights between neurons and save overall computation. The model structure used in Ross *et al.* [4] and Zhou *et al.* [100] is similar to AlexNet; that in Zhu *et al.* [8] is an improvement to VGGNet; while Zhu and Beroza [6] and Mousavi *et al.* [120] utilized the residual structure in ResNet.

Having over one million trainable parameters, Ross *et al.* [4] is neither a good candidate for training on small datasets nor a solution to fit an edge device with limited memory. Both Zhu and Beroza [6] and Zhu *et al.* [8] are smaller model that can be used for smaller regions. However, the need for an excellent event detector before conducting phase piking adds additional computation cost to Zhu and Beroza [6]. A separate RNN based event detector was proposed by the same group later Mousavi *et al.* [120]. Even though it works well for the deployment in the cloud, it significantly increases the overall model size. More importantly, both Zhu and Beroza [6] and Mousavi *et al.* [120] are trained given a fixed time resolution of outputs. While it may result in the best overall accuracy, any change in that resolution results in an entirely retraining of the entire model for both detection and picking algorithms. This dependency to predefined resolution largely limits their applications in the deployment on an embedded system, where hardware constraints vary from device

to device and output resolution needs to be traded off vs. computation cost. Zhu *et al.* [8], on the other hand, depends on a moving-window structure to set the output resolution dynamically during deployment. The small CNN model in Zhu *et al.* [8] is repeated applied to each fixed-length window of seismic waveforms, and the prediction is drawn independently on each window. Coarsely sampled windows (for example, every 2 s) can be used for initial event detection. Once the detection is confirmed, finer windows (for example, every 0.2 s) are used to construct the CFs for P-wave, S-wave, and noise. The device can actively manage the window frequency depends on the available resource on the device, making it extremely flexible for real-time applications. Last but not least, Zhu *et al.* [8] achieves both event detection and phase picking using single CNN model making it more favorable should any real-time fine-tuning of the model are considered in the future.

The method presented in Zhu *et al.* [8] is adopted for this study for the following reasons:

- CPIC model has relatively simple structure and fewer trainable parameters;
- Event detection and phase picking shares the same CNN model;
- Generalizes well to regions that it was not trained on.

Since the shallow neural network in Ross *et al.* [4] has already achieved over 99% detection accuracy on a large training set, there is little need to use CNN models with too many layers. Thus, the residual structures in ResNet are not necessary and the VGGNet is sufficient and relatively simpler than ResNet. In addition, it is known that VGGNet can be better simplified than ResNet by techniques such as network pruning [121]. In fact, the summation operator and concatenation operator at the end of each blocks of ResNet and Inception models making the quantization scheme in Chapter 3 more challenging. Moreover, Zhu *et al.* [8] modified the original VGGNet structure in two places. VGGNet was long criticized to have too many parameters due to the extensive use of many FC layer layers. Zhu *et al.* [8] replace the FC layer layers by a series of CONV layer layers which reduced the number

of parameters by a factor of three. Zhu *et al.* [8] also adopt larger kernel size at the early stage of the CONV layer layers which is commonly seen in modern CNNs, such as Inception V4 [64]. The shared model structure in Zhu *et al.* [8] is preferred since only one set of weights needs for both event detection and phase picking. When confirmed arrival times are sent back to the edge device, only one model instead of two needs to be fine-tuned for additional labels. Zhu *et al.* [8] also demonstrated that the pre-trained model on one region can be transferred to a different region with fair accuracy, which can be easily improved via a fine-tuning process with additional labels on the local regions. This gives us a feasible scheme to deploy an initial model and then iteratively improve it while actively recording seismic waveform in the monitoring region.

CHAPTER 3

DEPLOY NEURAL NETWORK ON EMBEDDED DEVICES

The deployment of the neural network model involves many aspects: the model needs to be simplified to fit the computing hardware constraints of the chosen device; parameter quantization within the NN model is required to use the number representation of the hardware; firmware and hardware acceleration are necessary for speeding up the model inference; and communication ports are needed for transmitting data between the cloud server and edge devices, or among edge devices. In this chapter, the first two aspects are studied in detail, while the last two hardware-related aspects are discussed in general with details deferred to future work. Section 3.1 gives a summary of the seismic acquisition devices commonly used in field deployments today. Detailed information about implementing CNN models on embedded platforms is provided to explain why these devices can support an accurate, responsive, and reliable seismic monitoring system. Section 3.3 tries to find a way to improve the CNN model demonstrated in Chapter 2 by visualizing filter weights within the CPIC model. The computational burden of the CNN is alleviated by model simplification techniques presented in Section 3.4. The simplified model is then quantized in Section 3.5 with a fixed-point representation that is likely to be used on many seismic edge devices, which further reduces the memory cost.

3.1 Sensors for Seismic Monitoring

Seismic sensors are the devices detecting displacement, velocity, or acceleration of the ground motion resulting from fault movement or from transient oscillations as seismic waves pass under a monitoring site. Traditional monitoring systems typically provide only sparse observations as the receivers are spaced too widely or data are not continuous in time due to cost. Research-grade, high-resolution seismic sensors are expensive. In fact,

those specialty instruments are usually placed in long-term installations that are suitable for strong motions from large earthquakes. One of these stations costs tens of thousands of dollars to build and equip, including sensors, on-site data acquisition systems, telecommunications, and back-up power. However, critical seismic monitoring systems, such as earthquake early warning (EEW), that aim to detect and locate earthquakes in real-time will require long-term continuous monitoring with a densely populated array for better spatial resolution. Even the densest seismic networks, e.g., the Southern California Seismic Network (SCSN), typically do not have more than one sensor every ≈ 20 km, because these can cost hundreds of thousands of dollars, or more, each year to operate and maintain. The development of seismic monitoring systems in the last decade has been a pursuit of more densely observed earthquake ground motions, which has inspired improvements in both monitoring devices [122] and processing algorithms [123, 124].

Device and communication advances are opening up opportunities for very dense 1-D seismic sensor arrays with lower per-station costs. The oil and gas industry, which uses seismic sensors to conduct surveys of subsurface structures, has converted cabled sensor systems into wireless *nodal* instruments [125]. The nodal system eliminates the need for transport and installation of the heavy cables that once connected sensors together. Thus, these systems, which have batteries that last 30 or more days, are becoming attractive for use in earthquake studies. A network of 904 nodal nodes was deployed in a recent experiment on Mount St. Helens for two weeks and detected an order of magnitude more earthquakes under the active volcano compared to the traditional seismic network [126]. Weaker events that are unknown were detected using a six-month deployment of 5400 sensors over a ≈ 100 km area in Long Beach, California [78] (Figure 3.1). In Oklahoma, a nodal array is being used to track small earthquakes moving along previously undetected fault structures that are induced by wastewater injection during the horizontal drilling process [127, 128]. While nodal array systems enable the deployment of dense sensor arrays at lower costs than ever before, their recording time is limited to relatively short time periods

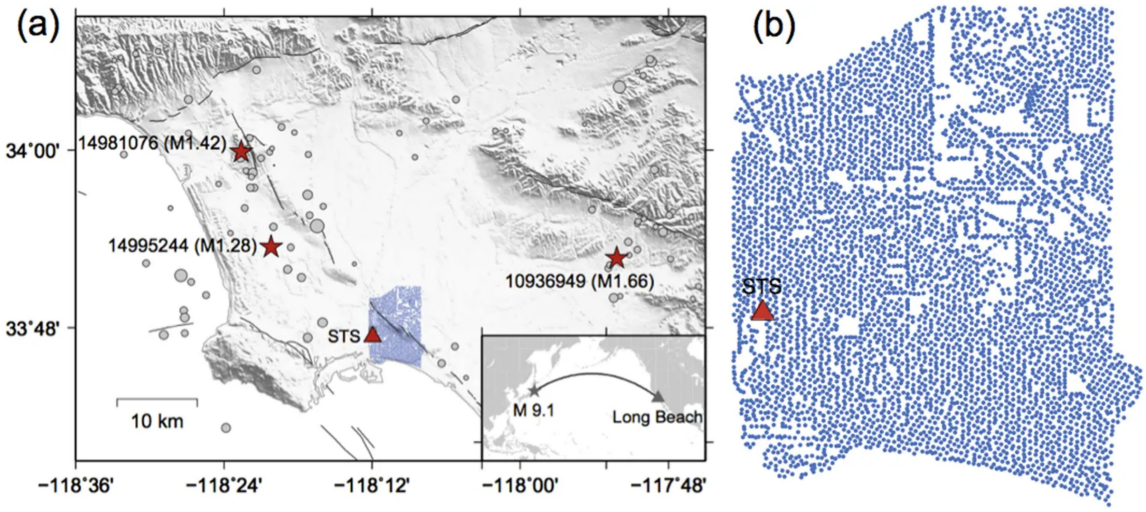


Figure 3.1: (a) Map of study region in Li *et al.* [78] for the Long Beach nodal array and local seismicity. (a) Blue dots are the 5200-sensor nodal array. A red triangle marks the broadband station STS belonging to Southern California Seismic Network (SCSN). Black curves denote the surface trace of mapped faults. Gray solid circles are seismicity listed in the SCSN catalog between January and June 2011, whose sizes are proportional to the magnitude. Red stars mark three selected cataloged events used for tests. The inset map shows locations of the M_w 9.1 Tohoku-Oki earthquake and its ray path to Long Beach. (b) Zoomed-in plot of the Long Beach array and the STS station.

(weeks to months) on a stand-alone battery.

Systems monitoring seismic activity over mobile devices, such as a smartphone, were developed for EEW purpose [129]. These low-cost, "personal seismometers" provide opportunities for long-term installations for wider coverage, especially in cities where it is difficult to deploy seismic sensors traditionally. These systems build on the availability of inexpensive low-power embedded processors with commercial-grade accelerometers as their seismic sensors. Accelerometers are ubiquitous in modern electronics, such as activating car airbags, rotating smartphone screens, and controlling gaming systems. Even though the sensors in these devices are of lower quality compared to scientific-grade instruments, they are adequate to capture moderate to strong ground motions and are readily available for broader coverage of large monitoring regions. Similar sensors have been installed in buildings to study the structural sway when a passing earthquake wave [130].

In the wake of embedded processing requirements of real-time seismic sensors, the

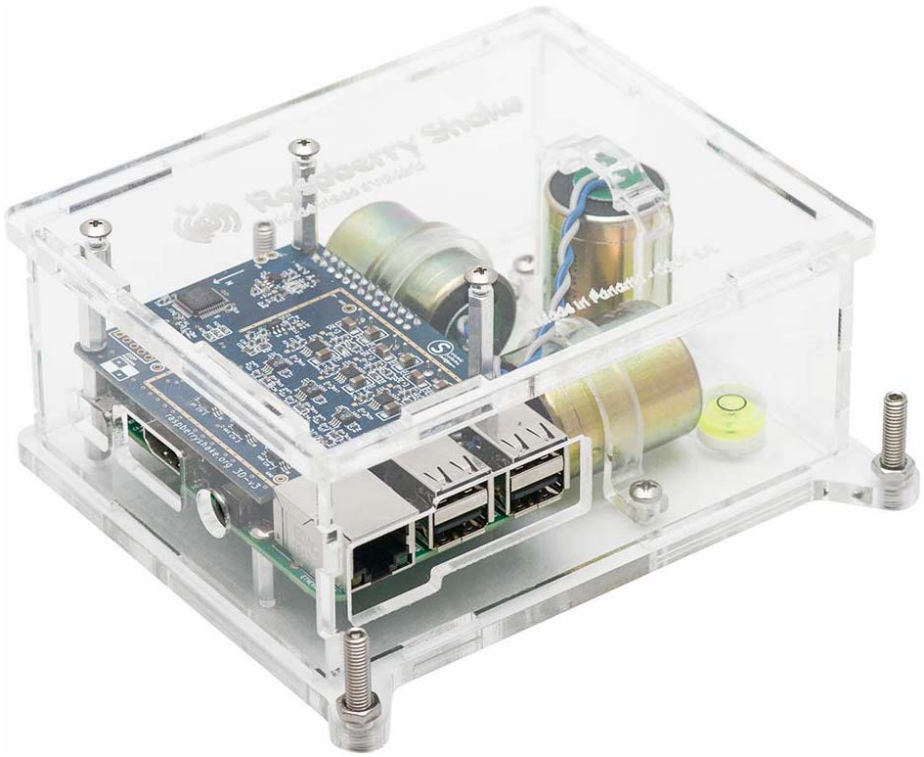


Figure 3.2: Raspberry Shake 3D consisting of Raspberry Pi computer plus three vertical geophones with 100 Hz sampling rate.

Raspberry Shake (Figure 3.2) devices were developed with a large variety of configurations containing typical seismic sensors. The onboard Raspberry Pi processor, which provides a four-core 1.4 GHz CPU and 1 GB of on-chip RAM, is a widely used embedded processing platform with extendibility potential to different types of sensor and communication modules [131]. Raspberry Shake sensors have been added to existing local seismic networks to complement the observations for moderate to large earthquakes, including in countries with more limited seismic monitoring infrastructure [132]. Anthony *et al.* [133] confirmed the effectiveness of the Raspberry Shake system on both field and laboratory observations. Different studies are being conducted to validate applications of the Raspberry Shake in many scenarios, including rockfall activity monitoring [134], as well as local [132], regional [135], and national seismic monitoring [136]. The Raspberry Shake has proven to be not only a good development platform but also a viable deployment system for real-time acquisition and processing of seismic waveforms. However, due to the limited computing

resource on most of the embedded processors, a pre-trained full NN model from a workstation cannot be directly deployed without further simplification.

3.2 Computation on Hardware

3.2.1 Convolution Strategies

The basis of a CNN model is the convolution operator between the input feature maps and the filter weights. Three different strategies have been proposed to implement a convolution operator, namely direct convolution, unrolling based convolution, and fast Fourier transformation (FFT) based convolution. The direct convolution computes the convolution via a sliding window of filter weights over the feature maps which computes their dot product. With proper padding and striding, the output can be formed into the desired shape. This approach is adopted by cuda-convnet2 [137], and Theano-legacy [138, 139]. However, this direct approach usually results in a rather implementation. Chetlur *et al.* [140] used a memory heavy, but more efficient way to compute the convolution operator. Each sliding window is unrolled into a row of a larger feature map matrix and the corresponding filter kernel is unrolled into a column of filter matrix. The convolution operation is then reduced to multiplying the filter matrix by the feature map matrix. Obviously, there are redundant entries in both matrices meaning that it uses more memory to store the intermediate values than the direct approach. On the other hand, the convolution operation involves moving a considerable amount of data in and out the processor which is usually the bottleneck instead of the actual computation. Converting the convolution to matrix multiplication gives a faster run time, because it performs a higher ratio of operations per byte of data transferred. This ratio increases as the matrices get larger, so we want to form the largest matrix multiplication that the memory supports. This unrolling based convolution approach is also adopted by many popular Frameworks including Caffe [141], Torch [142], Theano [139] and later inherited by TensorFlow [143] and PyTorch [117]. Vasilache *et al.* [144] proposed the FFT based scheme that converts the time-domain convolution into a product in

the Fourier domain. This conversion significantly reduces the computational complexity; however, the overhead of converting between the time domain and Fourier domain makes it less efficient for smaller filters. A detailed comparison of these three strategies is given by Li *et al.* [145]. Since the filter size in the CPIC model is either 3 or 5, the unrolling approach is preferred in this work.

3.2.2 Matrix-matrix multiplication

The computation for FC layers can be easily represented as a matrix-matrix multiplication operation. Since we adopt the unrolling approach in Section 3.2.1, the convolution operations for the CONV layers, which can be represented as a series of convolutional matrices, also consists of many matrix-matrix multiplications.

Indeed, consider a one layer in a CNN of the CPIC model as an example. There are M channels of inputs $x_m[n]$ for $m = 1, 2, \ldots, M$, each of length N, i.e., $n = 0, 1, \ldots N-1$. The CONV layer produces K channels of outputs $y_k[n]$ for $k = 1, 2, \ldots, K$, also of length N (prior to max pooling). Each input is connected to all outputs via 1-D convolution, where the filter length is L and the filter coefficients (called the impulse response in DSP) are $h_{k,m}[n]$ for $n = 0, 1, \ldots, L-1$. Thus we have MK filters – all with different filter coefficients. Altogether, there are MLK parameters needed to define all the filters. The k-th output channel, which is the sum of M filtered inputs, becomes

$$y_k[n] = \sum_{m=1}^{M} h_{k,m} * x_m = \sum_{m=1}^{M} \sum_{\ell=0}^{L-1} h_{k,m}[\ell]\, x_m[n-\ell] \qquad \text{for } n = 0, 1, \ldots, N-1 \quad (3.1)$$

where $*$ denotes 1-D convolution.

One-dimensional convolution for one output channel can be expressed as a matrix-vector multiplication. Since the length of the filters is much less than the input signal lengths, it is best to create the convolution matrices from the M input channels. The result is an $(N + L - 1 \times L)$ matrix $\boldsymbol{X}_m^{\text{Full}}$, where each column of $\boldsymbol{X}_m^{\text{Full}}$ contains all the signal

values in the m-th input channel. Successive columns are staggered by the stride length. When the stride is one, the result is a Toeplitz matrix. Here is an example for $N = 7$ and $L = 3$ (the subscript m is dropped from x_m), where the matrix is $(N + L - 1) \times L = 9 \times 3$:

$$\boldsymbol{X}_m^{\text{Full}} = \begin{bmatrix} x[0] & 0 & 0 \\ x[1] & x[0] & 0 \\ x[2] & x[1] & x[0] \\ x[3] & x[2] & x[1] \\ x[4] & x[3] & x[2] \\ x[5] & x[4] & x[3] \\ x[6] & x[5] & x[4] \\ 0 & x[6] & x[5] \\ 0 & 0 & x[6] \end{bmatrix} \qquad \text{Full Convolution matrix is } 9 \times 3$$

If we want the output channels to have the same length as the input channels, we must remove two rows — conventionally the first and last, or the last two, and then we call the resulting matrix $\boldsymbol{X}_m$. To get the matrix-vector form of convolution, we now define a length-N column vector for the k^{th} channel of output, $y_k[n]$, as

$$\boldsymbol{y}_k = \begin{bmatrix} y_k[0] & y_k[1] & \dots & y_k[N-1] \end{bmatrix}^T$$

and a length-L column vector for a filter between m^{th} input channel and k^{th} output channel

$$\boldsymbol{h}_{k,m} = \begin{bmatrix} h_{k,m}[0] & h_{k,m}[1] & \dots & h_{k,m}[L-1] \end{bmatrix}^T$$

The matrix-vector form for the convolution in (3.1) becomes

$$\boldsymbol{y}_k = \sum_{m=1}^{M} \boldsymbol{X}_m \boldsymbol{h}_{k,m} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \dots & \boldsymbol{X}_M \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{h}_{k,1} \\ \boldsymbol{h}_{k,2} \\ \vdots \\ \boldsymbol{h}_{k,M} \end{bmatrix}}_{\text{kernel}}$$

Now we put all convolution matrices into one matrix $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \dots & \boldsymbol{X}_M \end{bmatrix}$ and define the *kernel* for the k^{th} output as a collection of filters, $\boldsymbol{h}_k = \begin{bmatrix} \boldsymbol{h}_{k,1}^T & \boldsymbol{h}_{k,2}^T & \dots & \boldsymbol{h}_{k,M}^T \end{bmatrix}^T$. Finally, we write one matrix product for all the outputs of this single CONV layer as

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_K \end{bmatrix} = \boldsymbol{X} \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_K \end{bmatrix} = \boldsymbol{X}\boldsymbol{H} \tag{3.2}$$

where $\boldsymbol{Y}$ is $N \times K$, $\boldsymbol{X}$ is $N \times ML$, and $\boldsymbol{H}$ is $LM \times K$.

Figure 3.3 gives an example of such an unrolled convolution between a 3×6 input feature map (X_{ij}) and two three-channel convolution kernels (H_{ij} and G_{ij}). Convolution matrices of the feature map with full zero-padding are constructed for each input channel (represented in three different colors) and concatenated horizontally into one **input matrix**. Since the stride value in this example is 1, notice that the matrix structure inside each colored block of the **input matri**x is Toeplitz. If a stride value larger than 1 is used, some of input matrix rows are removed for the corresponding windows skipped due to the larger stride value. On the right hand side, the filter weights of different input channels are concatenated vertically into columns of the **kernel matrix** according to the channel order in the input matrix (using the same color code). Different filters are stacked horizontally to so that their outputs can be computed simultaneously. Note that the kernel matrix is invariant to the specific stride choice. Thus, the computation of the output (yellow) for this CONV layer can be represented as a multiplication between the input and kernel matrices.

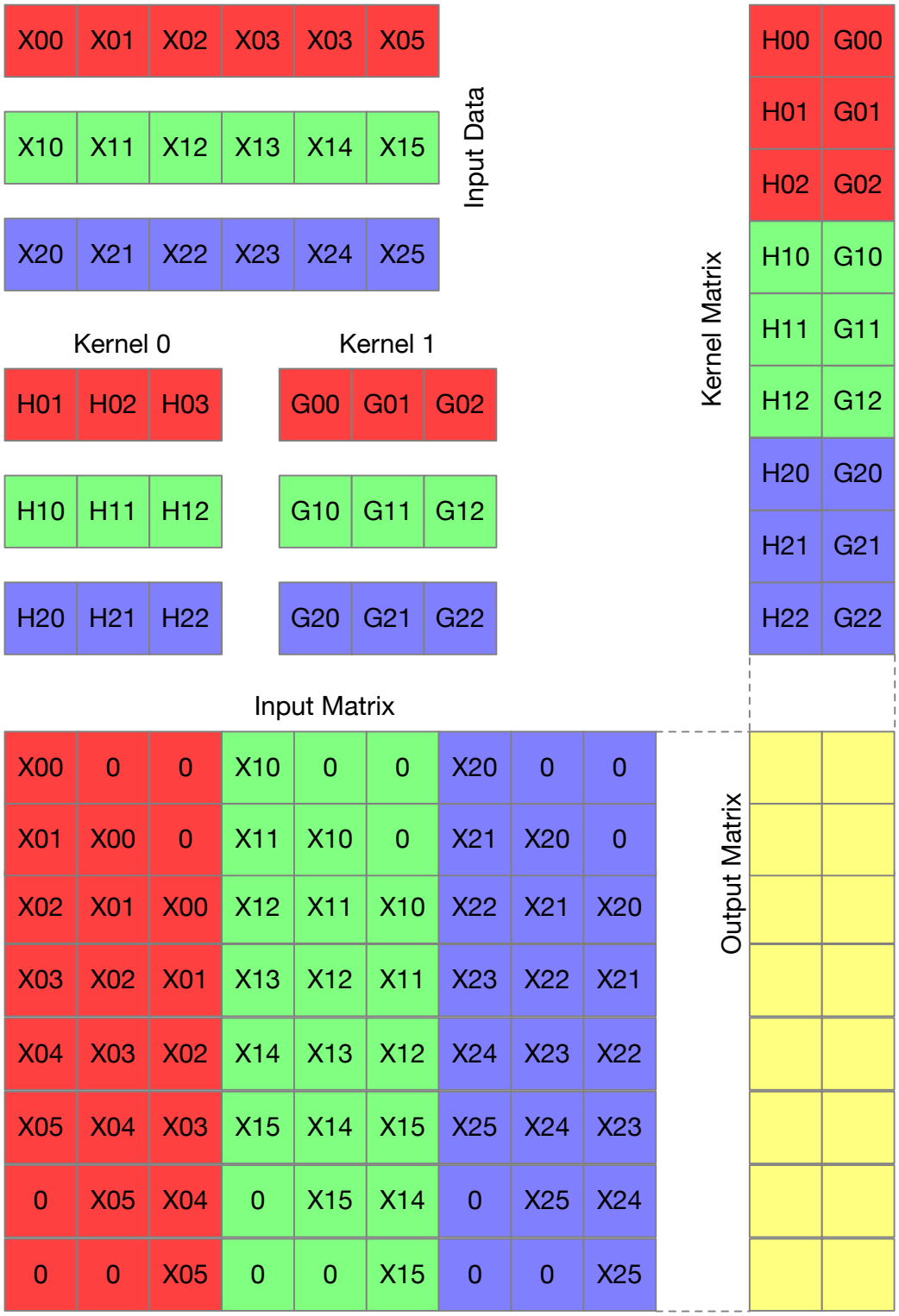


Figure 3.3: Convolution example (with zero-padding) between a three input channels and two size-9 kernels. Notice that $M = 3$, $N = 6$, $L = 3$, and $K = 2$ as defined in (3.1).

As a result, accelerating the computation of matrix-matrix multiplication (e.g., $\boldsymbol{C} = \boldsymbol{AB}$) can significantly improve the computation of the overall CPIC model, which relies on CONV layers and one FC layer. Such operations are conventionally performed by a function called general matrix multiply (GEMM). Most GEMM functions from the basic linear algebra subprograms (BLAS) are optimized for scientific computing where matrices might be as large as thousands of double-precision floating-point elements. Thus, these GEMM functions typically repack two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into smaller shapes as shown in Figure 3.4 when the overall matrix size exceeds the L1 cache size of the microkernels. Each pair of these smaller matrices is fed into a microkernel to compute their product and accumulated outside for the final results. The bottom line is using fewer filters in each kernel, or fewer kernels, can significantly reduce the overall computation cost.

Another issue is number representation. When fixed-point numbers are used for the seismic acquisition system, it is natural to use fixed-point numbers for the CNN computation as well. However, when using the same GEMM functions, accumulation of fixed-point numbers outside the microkernel might be necessary but is very undesirable. Instead, using lower-precision fixed-point numbers is preferred so that the entire dot product can be computed at once inside a single microkernel at high precision as shown in Figure 3.4. Then the dot product result is shifted back into a compatible lower-precision fixed-point number as the output of the microkernel. This puts an additional constraint on the system design that the memory required for each dot product block needs to be less than the L1 cache size of the processor. Note that most embedded processors, including the Raspberry Pi, have more than 16 KB of L1 cache. This allows multiple outputs to be computed inside each microkernel as shown in Figure 3.4 using lower-precision numbers, which further improves the computation efficiency.

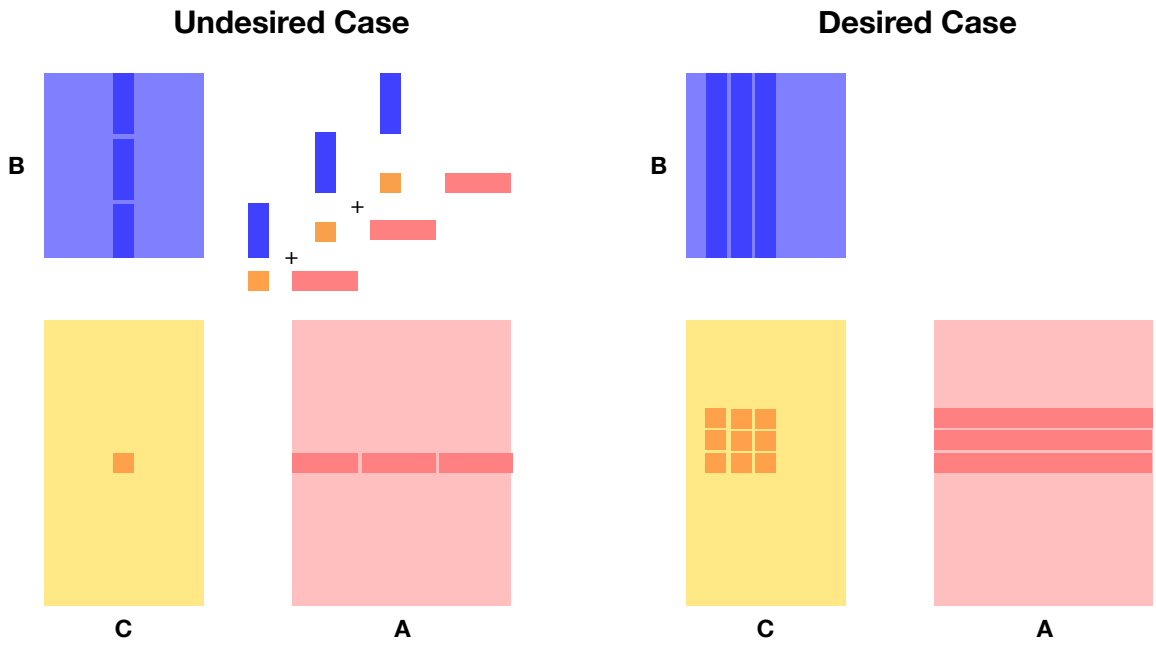


Figure 3.4: GEMM implementations for (undesired case, left) high-precision floating-point system where rows and columns must be broken up to compute one output of the large matrix, and (desired case, right) low-precision fixed-point system where entire rows and columns are available in the L1 cache to create small submatrices of the output.

3.3 Visualizing CNN filter weights

To simplify the CPIC model, two approaches are studied in this section. First, by visualizing the filter weights, redundant filters can be identified and removed. After such pruning the entire CNN model must be fine-tuned and revalidated to make sure that the desired classification accuracy is maintained on the simplified model. Second, the distribution of filter weights on each CONV layer spans a relatively consistent dynamic range. This property allows us to use a lower-precision fixed-point number representation, because the small errors in number representation should not change the classification outputs very much. Both of these phenomena can be exploited to further simplify the CNN model complexity, as well as reduce the memory cost.

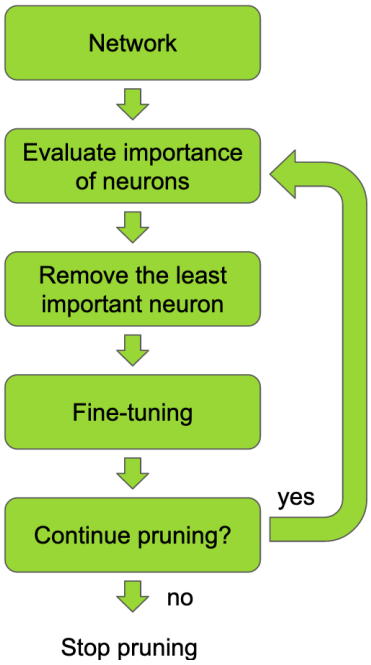


Figure 3.5: Flow diagram of neural network pruning as a backward filter presented as Figure 1 in Molchanov *et al.* [121].

3.3.1 CNN Filter Weights Close to Zero

Although there are many filters within each layer of the original CPIC model, there exist some filters most of whose weights are very close to zero. These filters can be found by visualizing the magnitude of each kernel's filter weights as a 2-D image. Eliminating such kernels helps to reduce unnecessary computation and memory usage while having little effect on the final classification results. This process is known as network pruning.

The general steps of network pruning [146] are shown in Figure 3.5: the importance of each neuron is evaluated by looking at the effect of its filter weights. A computationally costly scheme of ranking the importance of each neuron has been proposed by Molchanov *et al.* [121] from Nvidia; a similar approach was proposed by Anwar *et al.* [147] as well. These techniques prune the network by running the following optimization program:

$$\min_{\mathcal{W}'} \left| \mathcal{C}(\mathcal{D}|\mathcal{W}') - \mathcal{C}(\mathcal{D}|\mathcal{W}) \right|, \text{ s.t. } \|\mathcal{W}'\|_0 \leq B \tag{3.3}$$

where $\mathcal{W}$ is the filter weights, $\mathcal{C}$ is the final classification accuracy, $\mathcal{D}$ is the training data, and B is a positive integer. Although (3.3) is the ultimate goal one would want for network pruning, solving it requires exhaustively trying out all possible combinations of removing different redundant weights. Such a solver requires a massive amount of computing power that is not commonly available except at those organizations with large computing clusters having thousands of GPUs, such as Nvidia. More importantly, the pruned network is optimized for a fixed set of data and fine-tuning of the pruned model would require repeated runs of the optimization program in (3.3). This is not possible on any existing edge device, nor preferable, since the model on the edge needs to be self-adaptive to new incoming waveforms at a reasonable computing cost.

Instead of looking at individual filter weights, each filter and kernel must be treated as a whole and kept or removed altogether. By visualizing filter weights and looking for kernels whose weights are mostly close to zero, the kernels with most of their weights close to zero are identified. By removing those kernels, the minimum number of kernels are determined for each layer. The absolute value of the filter weights within CPIC layers are shown in Figures 3.6 – 3.8. At the i-th layer, the M_i filters connected to one output make up the kernel for that output. Each kernel is displayed as a rectanguar image, and one row of the image is one of the filters within a kernel. All kernels within the same layer are shown side by side in Figures 3.6 – 3.8. Note that filter weights are hard-clipped at 10 to highlight the difference between small and large filter weights. Yellow and cyan regions correspond to weights with large magnitudes, while the black regions correspond to those with close to zero magnitudes. Since there are only a limited number of kernels (K_i) in each layer, the number of kernels having all weights close to zero is easy to count. In Figure 3.6, nine filters in layer #1 and 13 filters in layer #2 are close to zero. This is a clear indication that the CPIC model can be simplified by removing those filters with small weights.

Unfortunately, the percentage of close-to-zero kernels is less on the deeper layers. Figure 3.7 shows the weights on layer #4, layer #7, and layer #11, respectively. Even though

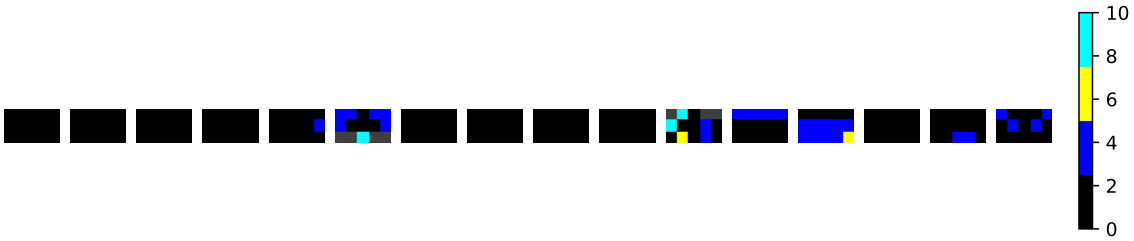
Table 3.1: Number of kernels in each layer of the CPIC model during every iteration of the network pruning process.

Layer	1	2	3	4	5	6	7	8	9	10	11	fc
Original number of kernels	16	32	64	64	64	64	64	64	64	64	64	64
# close-to-zero kernels	9	13	25	26	28	22	30	30	30	31	44	24
1st iteration number of kernels	8	16	32	32	32	32	32	32	32	32	16	32
2nd iteration number of kernels	8	16	32	32	32	16	16	16	16	16	0	32
3rd iteration number of kernels	8	16	32	32	32	16	16	8	8	8	0	32
4th iteration number of kernels	8	16	16	32	16	8	8	4	4	0	0	16
5th iteration number of kernels	8	16	16	32	32	16	16	8	8	0	0	32

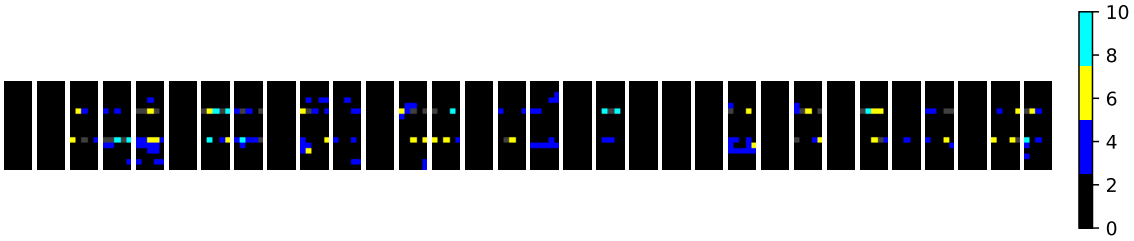
there are still kernels on layers #4 and #7 whose weights are almost all close to zero, many of them tend to have a small number of large weights, with the rest close to zero. On layer #11, the situation becomes similar to the first two layers where a majority of the kernels (56 out of 64) have all their weights close to zero.

In Figure 3.8, the weights on the FC layer are shown with dynamic ranges clipped between 0 and 1. With 64 inputs and 3 outputs, some of the weights on this FC layer are clearly close to zero. When a column of this image is all zero, that input feature doesn't matter for classification. For Figure 3.8, this indicates that the final classifier made its decision on a small subset of the features in the total 64-element feature space. Thus, it is likely that the CPIC model can be further simplified by using a smaller feature space for classification.

This pruning process can be conducted multiple times to further reduce the overall number of kernels (and individual filters). Empirically, the pruned model achieves better classification results by fine-tuning on a series of models whose kernels are gradually removed instead of a single model with all redundant kernels removed at once. Table 3.1 records the number of kernels on each layer during five pruning iterations. During the second and fourth iterations, the number of kernels on the last CONV layer is small enough that we merge it with the previous layer. This reduces the overall number of layers. After the fourth pruning iteration, the resulting model experienced a severe drop in the classi-



(a) Layer #1: 9 (out of 16) kernels close to zero at $k = 1, 2, 3, 4, 7, 8, 9, 10, 14$



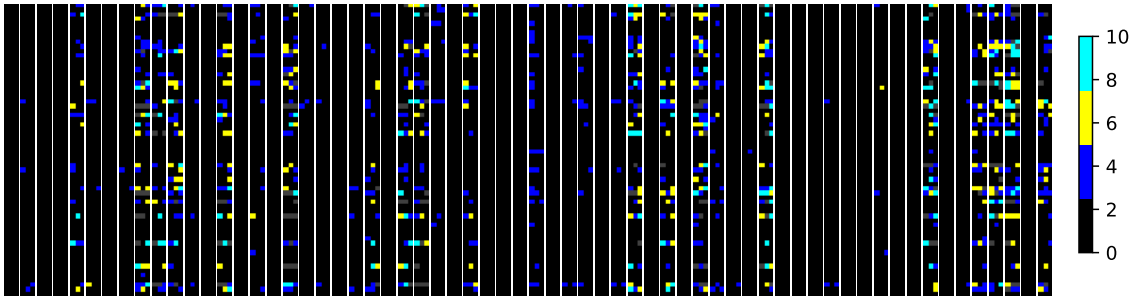
(b) Layer #2: 13 (out of 32) kernels close to zero
at $k = 1, 2, 5, 8, 11, 14, 17, 19, 20, 21, 23, 25, 26, 30$

Figure 3.6: Absolute values of the filter weights in the first two layers of the CPIC model which use 5-point convolutions. For each subimage one row is a filter, so the horizontal dimension is $L_i = 5$, and the vertical dimension is M_i, the number of layer inputs, 3 for layer #1 and 16 for layer #2. The number of subimages is equal to the number of layer outputs, $K_1 = 16$ for layer #1 and $K_2 = 32$ for layer #2. See Figure 2.2 in Chapter 2 for a diagram of the entire CNN.

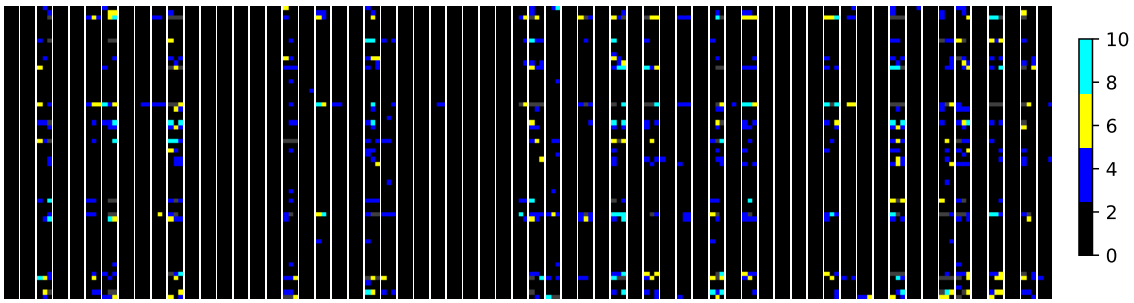
fication accuracy. We suspect that the model complexity was reduced too much, and the simplified model was unable to handle the variance inside the training dataset. Thus, we add back additional kernels in the fifth iteration to increase the model complexity and ultimately decided to use nine CONV layers in the final model.

3.3.2 Distribution of the CNN Filter Weights

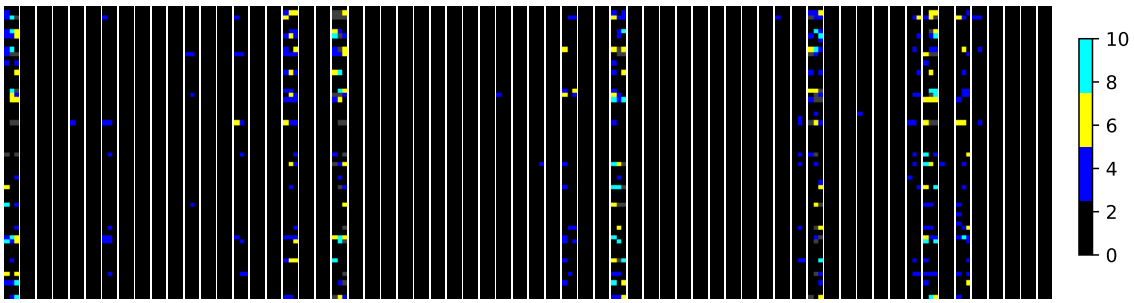
In addition to the absolute value of the filter weights, the distribution of these weights on each layer must also be considered. Distributions of the weight in all 11 CONV layers, as well as the final FC layer are shown in Figure 3.9 with 20 bins and a constant range from -10 to $+10$. Except for the first CONV layer, the weight distributions on all the rest of the layers resemble each other. More importantly, for all the layers these distributions are symmetric and tightly centered around zero. This phenomenon allows us to use a fixed-



(a) Layer #4: 26 (out of 64) kernels close to zero
for $k = 1, 3, 4, 7, 13, 15, 17, 21, 24, 28, 30, 31, 32, 34, 37, 40, 42, 48, 49, 50, 52, 53, 55, 56, 58, 63$



(b) Layer #7: 30 (out of 64) kernels close to zero
for $k = 1, 2, 4, 5, 8, 12, 13, 14, 15, 16, 17, 22, 25, 26, 28, 29,$
$30, 31, 35, 37, 39, 43, 47, 48, 49, 50, 54, 57, 60, 62$



(c) Layer #11: 44 (out of 64) kernels close to zero;
nonzero for $k = 1, 5, 7, 12, 15, 18, 21, 31, 33, 35, 38, 48, 49, 50, 53, 56, 57, 58, 59, 60$

Figure 3.7: Absolute value of the filter weights in CONV layers with length-3 filters where $L_i = 3,\ M_i = 64,\ K_i = 64$: (a) layer #4 (b) layer #7; and (c) layer #11. At least half of the kernels in most layers are close to zero. The number of kernels close to zero (black) increases, and then decreases, in the deeper layers.

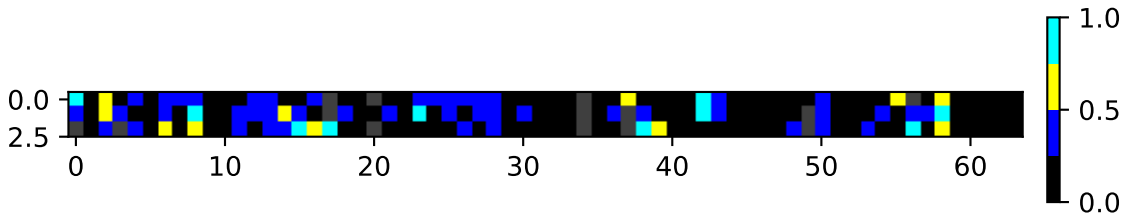


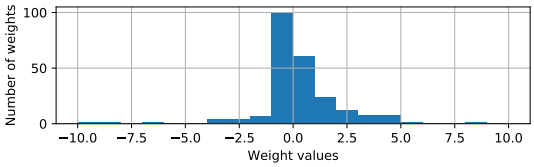
Figure 3.8: Filter weights in the FC layer of CPIC model with 64 inputs and 3 outputs. Each column represents the three weights from one input neuron to all three output neurons. Approximately 30 of these columns have all weights close to zero. Note that the dynamic range of these images is reduced to lie between 0 and 2.

point number system to represent the weights in lower precision and further reduces the memory, as well as the computation cost of the CPIC model. Furthermore, a symmetric dynamic range of the fixed-point number can be adopted without a bias term due to the symmetry around zero in all the filter weight distributions. On the other hand, there is no need to consider the bias term for the feature map distribution. Since each CONV layer is followed by a ReLU activation function, the resulting feature maps are also nonnegative which allows us to use unsigned fixed-point numbers to represent the feature map values.

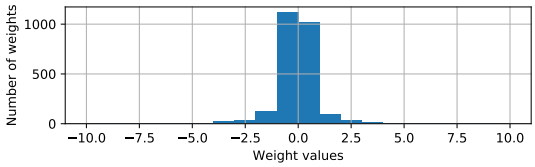
3.4 Model Simplification

3.4.1 Reducing Network Depth

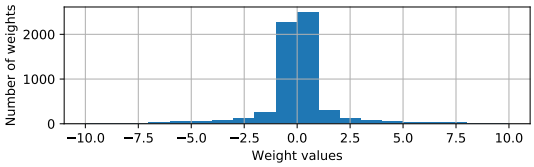
Since each CONV layer is followed by a max-pooling layer of size two, the size of the feature map in the time dimension is reduced by two following each CONV layer. Thus, the overall downsampling ratio is controlled by the number of CONV layers. In order to reduce the model complexity by removing some of the CONV layers, the duration of the input window could be shortened to have fewer time samples in the 3-C waveforms. For different window lengths T_w, the labeled phase arrival time must always be positioned at $0.25T_w$ from the beginning. As shown in Table 3.2, some other possible window lengths were tested, even a longer window, but a 20-s window duration works best for the Wenchuan dataset. Recall that for each manually picked phase, CPIC uses a 20-s long window starting



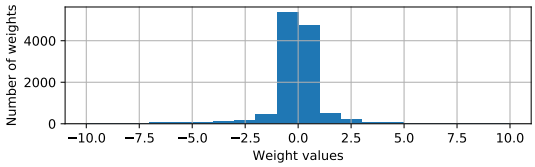
(a) CONV layer #1



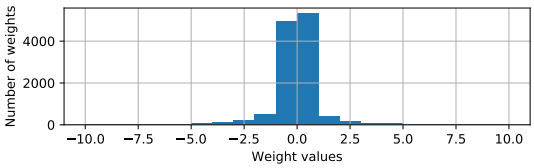
(b) CONV layer #2



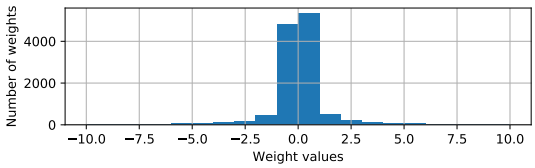
(c) CONV layer #3



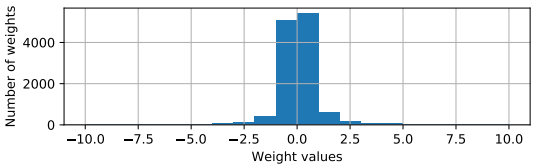
(d) CONV layer #4



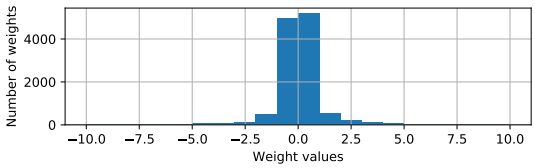
(e) CONV layer #5



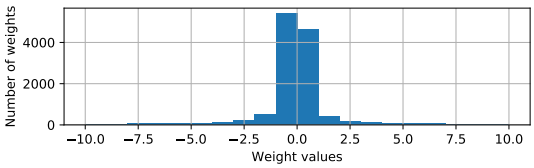
(f) CONV layer #6



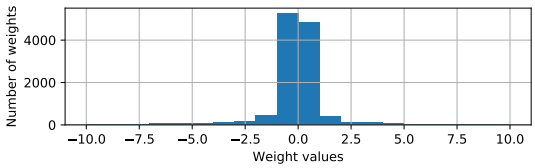
(g) CONV layer #7



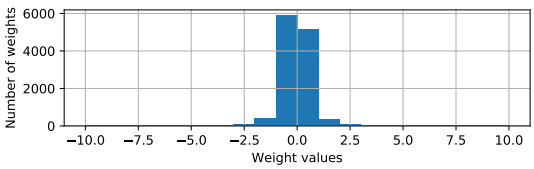
(h) CONV layer #8



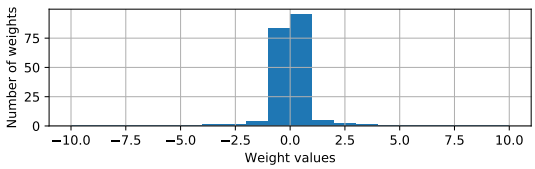
(i) CONV layer #9



(j) CONV layer #10



(k) CONV layer #11



(l) FC layer

Figure 3.9: Distribution of filter weights on every layer of the CPIC model before pruning. The total number of weights is $L_i M_i K_i$ for the i-th layer. Consult Table 3.4 for the specific values of L_i, M_i and K_i.

Table 3.2: Classifier accuracy (defined in (2.5)) vs. window lengths.

Window Length (sec)	2.5	5	10	20	40
Accuracy(%)	94.7	96.3	96.9	97.4	97.2

5 s before the pick and ending 15 s after as one window of a seismic phase (Figure 2.5). Thus, removing layers by shortening the input window length is not a right approach.

Another approach is possible by examining the FC layer. Since the feature space sizes in the CONV layers and the FC layer have some redundancy as shown in Figures 3.7 and 3.8, the trade-off between more CONV layers vs. a larger FC layer discussed in Section 2.1.2 can be reconsidered. In fact, the last few CONV layers perform convolutions between a very short input signal and a small kernel which is nearly the same as a fully connected structure, so it is reasonable that these could be merged into the final FC layer. Specifically, we can gradually reduce the feature space size from 64 to 4 until the max-pooling layer reduces the number of time samples from 2,000 to 8. This results in an 8×4 feature tensor which can be flattened into a 32×1 feature vector for the final FC layer.

3.4.2 Reducing Feature Space Size

In Section 3.3, it is shown that all CONV layers and the final FC layer have some redundancy in the filter weights. Thus, the overall number of filters on each layer, or the feature space size, can be reduced. To present tweaking of an individual layer and demonstrate the overall effect on the model classification accuracy, feature space sizes are reduced simultaneously with a fixed ratio on all layers. Table 3.3 summarizes the CNN model classification accuracy, given different levels of reduction in the feature space size on all layers. The original CPIC model has a feature space size of 64 and reaches a classification accuracy of 97.4%. By reducing the feature space size to 32, the loss in classification accuracy is negligible (only 0.1%). However, further reduction in the feature space size to 16 results in a noticeable drop in the classification accuracy, although it is still within 1% of the original model. When the feature space size is reduced to 8, a drop of a little more than 1% in

Table 3.3: Classification accuracy (defined in (2.5)) vs. the final feature space size.

Feature Space Size	8	16	32	64
Number of parameters	1972	7,320	30,544	107,248
Accuracy (%)	96.3	96.8	97.3	97.4

classification accuracy is observed. This serves as a guideline in simplifying the CNN for improving the overall CPIC design.

3.4.3 Simplified Model

Combining the knowledge we gained from reducing model depth in Section 3.4.1 and reducing feature space size in Section 3.4.2, we were able to further simplify the original CPIC model as shown in Figure 3.10. The overall number of parameters decreases from 107,440 in the original CPIC model (Table 3.4) to 9,112 in the simplified model (Table 3.6), almost a factor of 12 reduction. Following the observation in Section 3.4.2, the most significant feature space size has been reduced from 64 to 32, which results in about a factor of three reduction in the number of parameters. Based on the number of close-to-zero filters in the individual layers from Section 3.3, the numbers of filters on the deeper layers are no longer homogeneous. Rather it decreases along with the time domain which results in a further reduction in the number of parameters. The final FC layer is also replaced by a 32 to 3 perceptron node that reduces the number of parameters by half in that layer. Table 3.5 shows the simplified model after the first pruning iteration as described in Section 3.3.1. As demonstrated in Table 3.1, Figure 3.6 shows the structure of the final simplified model after five iterations. In addition, the depth of the network is also reduced from 11 to 9 based on the observation in Section 3.4.1.

These simplifications in deeper layers usually result in a more significant reduction in the overall number of parameters since they usually have a larger feature space size. The CPIC model, in particular, does not need a large number of features in the deeper layers which helps to limit the number of parameters further. On the other hand, simplification

Table 3.4: Original CNN architecture with 11 convolutional layers and one fully-connected layer (107,440 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×16	$2000 \rightarrow 1000$	240
1000×16	5×32	$1000 \rightarrow 500$	2560
500×32	3×64	$500 \rightarrow 250$	6144
250×64	3×64	$250 \rightarrow 128$	12288
128×64	3×64	$128 \rightarrow 64$	12288
64×64	3×64	$64 \rightarrow 32$	12288
32×64	3×64	$32 \rightarrow 16$	12288
16×64	3×64	$16 \rightarrow 8$	12288
8×64	3×64	$8 \rightarrow 4$	12288
4×64	3×64	$4 \rightarrow 2$	12288
2×64	3×64	$2 \rightarrow 1$	12288
FC-64 (64 inputs, 3 outputs)			
softmax			

Table 3.5: Simplified CNN architecture after first pruning iteration with 11 convolutional layers and one fully-connected layer (25,432 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×8	$2000 \rightarrow 1000$	120
1000×8	5×16	$1000 \rightarrow 500$	640
500×16	3×32	$500 \rightarrow 250$	1536
250×32	3×32	$250 \rightarrow 128$	3072
128×32	3×32	$128 \rightarrow 64$	3072
64×32	3×32	$64 \rightarrow 32$	3072
32×32	3×32	$32 \rightarrow 16$	3072
16×32	3×32	$16 \rightarrow 8$	3072
8×32	3×32	$8 \rightarrow 4$	3072
4×32	3×32	$4 \rightarrow 2$	3072
2×32	3×16	$2 \rightarrow 2$	1536
FC-32 (32 inputs, 3 outputs)			
softmax			

Table 3.6: Simplified CNN architecture after final pruning iteration with nine convolutional layers and one fully-connected layer (9,112 parameters).

input $(N \times M)$	kernel $(L \times K)$	max pooling	#parameters
2000×3	5×8	$2000 \rightarrow 1000$	120
1000×8	5×16	$1000 \rightarrow 500$	640
500×16	3×16	$500 \rightarrow 250$	768
250×16	3×32	$250 \rightarrow 127$	1536
127×32	3×32	$127 \rightarrow 64$	3072
64×32	3×16	$64 \rightarrow 32$	1536
32×16	3×16	$32 \rightarrow 16$	768
16×16	3×8	$16 \rightarrow 8$	384
8×8	3×8	$8 \rightarrow 4$	192
FC-32 (32 inputs, 3 outputs)			
softmax			

in shallower layers reduces the overall computation cost while having little effect on the number of parameters. Since the data tensors in the early layers tend to have longer input length in the time domain, they require more computation to finish a convolution operation compared to a short input length. Thus, a reduction in the number of filters in the first two layers may not contribute much to the overall reduction in number of parameters, but it does helps significantly in alleviating the computational burden.

The training process of the simplified CPIC model is shown in Figure 3.11. Comparing to the training process of the original CPIC model in Section 2.5.2, the simplified model reaches nearly the same classification accuracy at 97.2% vs. 97.4% for the original model. However, the training process takes more epochs to converge, and the validation accuracy in the early epochs becomes quite volatile. The simplified model converges near the final stopping accuracy after more than 70 epochs, while the same happens for the original CPIC model within 40 epochs. The orange curves for training in Figure 3.11 resemble those in Figure 3.11, indicating a stable training process. However, the blue curves in Figure 3.11 oscillate more dramatically in Figure 3.11 than those in Figure 2.15. The simplified model has less redundancy and can be more sensitive to perturbations in the input data when its weights have not gotten close to the optimal region. Notice that there is no over-fitting observed, as expected, on the simplified model given a training set of the same size.

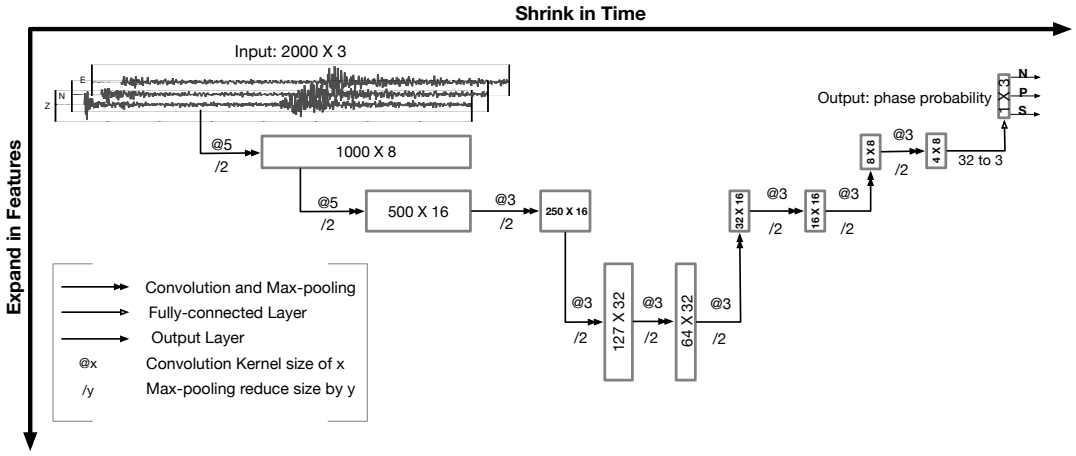


Figure 3.10: A diagram showing the data sizes in the simplified CNN network structure. Each input is a 3-C seismogram (20-s window) which shrinks in time but expands in the feature dimension as it passes through nine convolutional layers for feature extraction. The final layer is fully connected with three outputs that give the probabilities of a window being noise, P, and S phases.

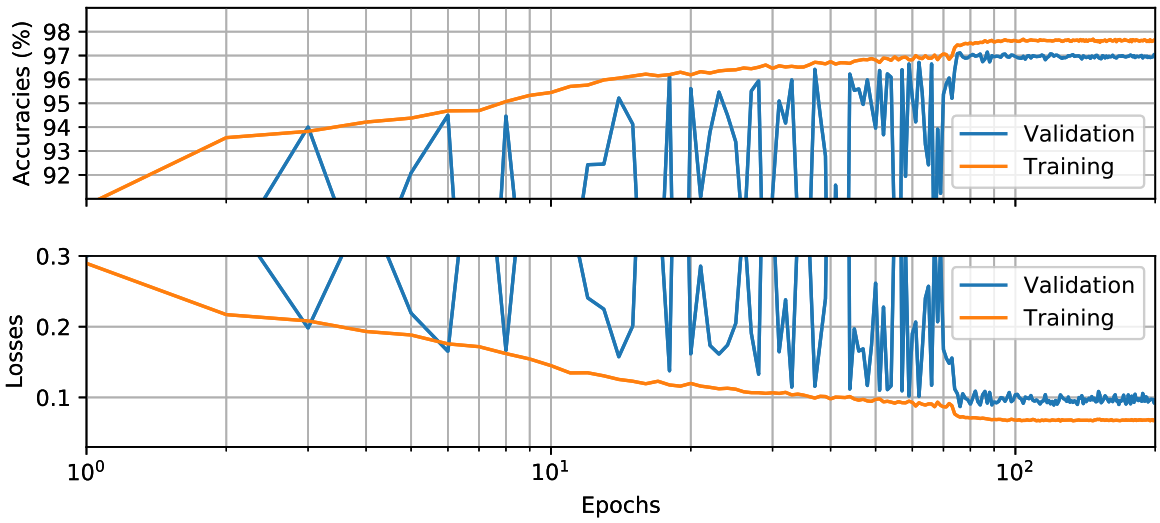


Figure 3.11: Training performance of simplified CPIC model on Wenchuan dataset: (a) classifier accuracy and (b) loss function against number of epochs for training and validation datasets during the CNN training process.

3.5 Quantization of CNN Model

As demonstrated in Section 3.3.2, the effective dynamic range of the weights in each layer of the CPIC model is relatively small. Furthermore, the seismic waveforms recorded on a seismometer are usually represented in a fixed-point (integer) format using a wordlength between 12 and 24 bits. It is natural to think about quantization of the CNN model as lowering the precision of arithmetic operations, as well as reducing both the amount of computation and the memory size. Instead of using either fixed-point or floating-point numbers, a dynamic fixed-point number has been proposed specifically for quantization in neural networks [148]. Arithmetic operations are conducted in fixed-point while keeping a separate scaling factor for each neural network layer as a floating-point number. A specialized simulation scheme, which can be run on floating-point only GPUs, is designed to validate the classification accuracy of the quantized CPIC model in dynamic fixed-point numbers. Then a fine-tuning process can be used to improve the classification accuracy when the CPIC model is quantized using fewer number of bits.

3.5.1 Finite Word Length Scheme

Summarized in Gevers and Li [149], the performance degradations due to finite-precision arithmetic in the filter implementation and computations are:

1. quantization of the input signals into a set of discrete levels and the ensuing quantization errors

2. overflow of the computations, which occurs when the computation results are out of the range of the fixed-point hardware capacity;

3. accumulation of roundoff errors that occur at arithmetic operations;

4. quantization of the filter coefficients into a finite number of bits and the ensuing quantization errors.

The first effect, input signal quantization error, depends on the ADC used on the seismic recording device. ADC errors are usually modeled as an additive uniformly distributed white noise, which has nothing to do with the filter structure or its parameters (filter coefficients). The other three effects are influenced by the particular implementation chosen for the filter. The overflow effect is based on the wordlength capacity of the computing system, which occurs when the dynamic range of the signals exceeds the maximum value that can be represent with a given wordlength. A proper scaling of the signals, which is a function of the signal dynamic range, is necessary to avoid serious errors caused by overflows.

The accumulation of roundoff errors after arithmetic operations and the errors induced by finite wordlength encoding of the filter weights are usually called Finite Word Length (FWL) effects. As soon as an ideal model (with full precision) is implemented in a FWL machine (an embedded device), some performance degradation due to FWL errors is inevitable. There are two basic ways of representing real numbers: fixed-point representation and floating-point representation. Internally, most computing system uses a binary number system. In *fixed-point binary representation*, the "two's complement" number representation is the most frequently used format for signed numbers. A real number x is represented in a quantized form as

$$Q[x] = K\left(-x_0 + \sum_{i=1}^{B} x_i 2^{-i}\right) \tag{3.4}$$

where K is an arbitrary scale factor and the x_i are binary digits which are either 0 or 1. Note that the first bit, x_0, controls the sign of x and is also known as the *sign bit*. If $x_0 = 0$, $0 \leq x \leq K$ is nonnegative; if $x_0 = 1$, $-K \leq x < 0$ is negative. The number inside the parentheses of (3.4) is between -1 and $+1$. Thus, any number with magnitude less than K can be uniquely represented by (3.4). When B is infinity, the representation is perfect; otherwise, when B is finite, we encounter a FWL effect.

The quantized representation is in the range of $-K \leq Q[x] \leq K - q$, where q is the smallest difference between any two of the total 2^{B+1} numbers, known as the *quantization step size*. Notice that the quantization error $|x - Q(x)| \leq q$ for any x, and is uniformly

distributed between $-q/2$ and $+q/2$ along the x-axis regardless of the magnitude of x. With proper choice of the scaling factor K in (3.4), the dynamic range of $Q[x]$ can be made identical to that of a floating-point number with the same number of bits. A fundamental difference is that the quantization error of fixed-point numbers is identical all through the dynamic range, while that of floating-point numbers is small for small numbers and large for large numbers. When the relative error is more relevant than the absolute one, which is often the case, floating-point representations typically deliver a better range – precision tradeoff than fixed-point ones. However, arithmetic operations with floating-point numbers are more complicated, so floating-point coprocessors are often added in modern day computers when computation speed is an essential factor.

The dataflow in the FC layer and the CONV layers consists many multiply and accumulate (MAC) operations as demonstrated by Figure 3.12. The feature maps are multiplied with the filter weights, and their results are accumulated to form the outputs. For example, a feature map of m bits and filter weights of n bits are multiplied and accumulated using an adder tree such as in Figure 3.12. The multiplication results in an $m + n$ bit output product, and the additions at each level of the tree add one more bit. In the last level, the bit-width is $m + n + \log_2(c)$, where c is the number of multiplications per output channel. After filtering, a bias is added to form the output which is then truncated to the lower precision representation by the shifters. Thus, our goal in quantization design is to find a good balance between reducing the bit-widths (m and n) and maintaining a good classification accuracy for the CNN. For simplicity, m and n are chosen to be the same in this study.

3.5.2 Inference using the Quantized CNN Model

To accommodate the large dynamic range difference in each layer of the neural network, a *dynamic fixed-point* scheme was proposed for neural networks [148]. It was proposed originally for faster training; however, Gysel *et al.* [151] later demonstrated that this dynamic fixed-point scheme has more value in approximating NN models during inference.

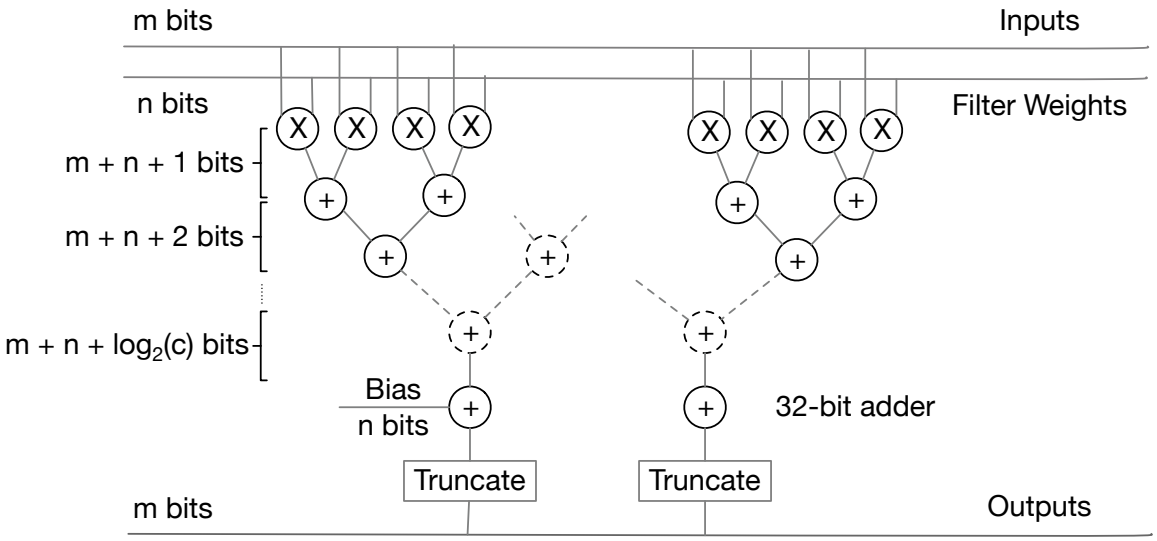


Figure 3.12: Fixed-point number operation in convolutional neural networks. Adapted from Figure 5.1 in Gysel [150].

Similar to a block floating-point number format, the dynamic fixed-point representation uses a scaling factor S^l, which is a power of 2, represented in floating-point for scaling each NN layer. Effectively, S^l determines the location of the binary point of each number in that layer and thus controls the the fractional length of the fixed-point number. The filter weights are still represented as fixed-point numbers using this shared scaling factor as in (3.5):

$$x^l = S^l \left(-x_0^l + \sum_{i=1}^{B} 2^{-i} \right) \tag{3.5}$$

Comparing with (3.4), each layer has a dynamically changing scale factor S^l instead of a global fixed scaling factor K. However, the same S^l is shared within a single NN layer, which is different from the exponent x_e in floating-point which can vary from weight to weight. More importantly, the evaluation of this shared scaling factor can be delayed until the last layer of each branch. Since the CPIC model only has one branch, there will be only one global scaling factor in floating-point at the end. During CNN inference, the filter outputs are computed in fixed-point, where only the relative values, instead of the absolute values are tracked. Since both ReLU and max-pooling, at the layer output, operate based

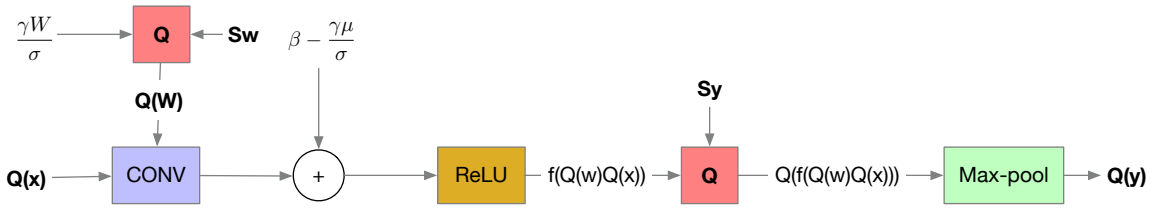


Figure 3.13: Flow diagram of a CONV layer quantized using the dynamic fixed-point number system. Notice that the batch norm has been absorbed into the filter weights W. There are two scaling factors: S_w for filter weights and S_y for feature maps.

on comparing numbers, relative values are sufficient.

Figure 3.13 shows the overall inference pipeline of the quantized CNN model. The input x and filter weights W are both quantized into a B-bit representation as $Q(x)$ and $Q(W)$ respectively. The CONV layer output will be a $4B$-bit fixed-point representation $Q(W)Q(x)$, which will be bottom clipped by ReLU. The precision of the nonlinearly activated data $f(Q(W)Q(x))$ will then be reduced to B bits before passing into the max-pooling layer. Notice that the Q block can be performed after the max-pooling layer; however, it is faster to find the max value in the fixed-point representation rather than in floating-point. There are two scaling factors, S_w and S_y, involved in the process. Combining those with the scaling factors S'_w and S'_y from the previous layer, denoted as $S_x = S'_w S'_y$, the global scaling factor can be computed as

$$S = S_0 \prod_{i=1}^{N} S_w^i S_y^i \tag{3.6}$$

where S_0 is the input scaling factor, S_w^i and S_y^i are the dynamic scaling factors for the i^{th} layer, and N is the number of layers.

The batch norm layers are omitted in Figure 3.13 since they are usually absorbed into the previous CONV layer. Indeed, the CONV layer operation

$$y = Wx + b \tag{3.7}$$

where W is the filter weights and b is the filter bias, can be combined with the batch normalization layer (BN layer) operation to obtain

$$y = \gamma \left(\frac{x - \mu}{\sigma} \right) + \beta \tag{3.8}$$

where μ is the sample mean, σ is the sample standard deviation in the current batch (a collection of training samples), γ is the scaling factor and β is the bias factor, both of which are learnable parameters in the training process. The result of combining (3.7) and (3.8) into a CONV layer operation is (3.9) below

$$y = \gamma \left(\frac{Wx + b - \mu}{\sigma} \right) + \beta = W'x + b' \tag{3.9}$$

where $W' = \frac{\gamma W}{\sigma}$ and $b' = \beta - \frac{\gamma \mu}{\sigma}$ are the filter weights and bias of the new CONV layer with the BN layer absorbed. This is also the reason that CONV layers followed by a BN layer do not need a bias in the filter coefficients.On the other hand, the BN layer dynamically estimate the sample mean and variance of each batch during the training process while using fixed values during validation. The dynamically estimated batch statistics are accumulated by an exponential moving average, i.e.,

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha Y_t + (1 - \alpha) S_{t-1}, & t > 1 \end{cases} \tag{3.10}$$

where Y_t is the estimator of the current batch, S_t is the accumulated estimator of all batches, and α is the exponential decay of the weighting.

Given enough number of bits, the direct quantization of most CNN models is sufficient to approximate the original floating-point model as observed in Krishnamoorthi [152]. However, a fine-tuning process can be developed to obtain improvements over the directly quantized models.

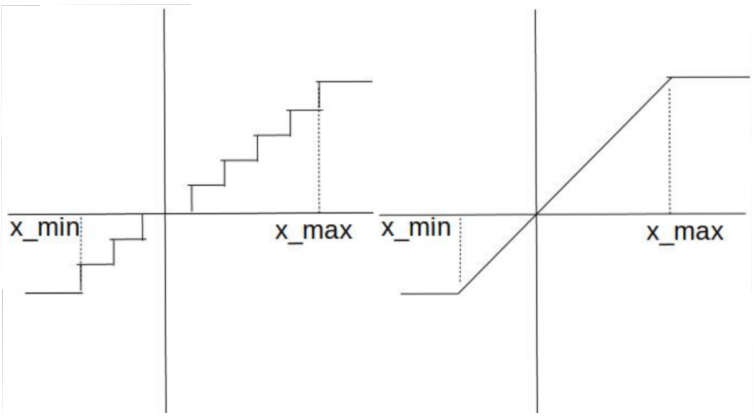


Figure 3.14: Simulated quantizer (left) and an approximation (right) for the purpose of calculating its derivative, from Figure 1 in Krishnamoorthi [152].

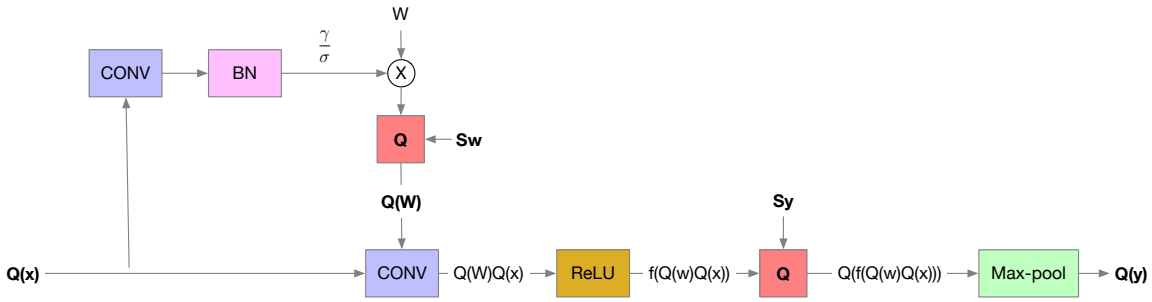
3.5.3 Fine-tuning the Quantized CNN Model

Fine-tuning of a quantized CPIC model is based on a simple assumption. The optimal quantized model should be similar enough to the original floating-point model that with small tweaking of filter weights, the overall classification accuracy can be recovered. Gradient-based approaches can still be used to fine-tune the quantized weights of the neural network; however, an approximation of the derivative of the quantizer operator is needed for the chain rule to pass the gradient during the backpropagation process. The quantizer is a nonlinear and non-differentiable function as shown in Figure 3.14 (left), but the piecewise linear approximation in Figure 3.14 (right) is a function commonly used as a proxy for the quantizer's derivative. The exact outputs of the nonlinear quantizer are used in the forward pass, while the approximated derivatives are used in the backward pass. This discrepancy between forward and backward passes should be negligible if the quantizer has enough number of bits. Indeed, as the number of bits in the quantizer increases, the step size decreases and the closer two functions in Figure 3.14 resemble each other. When an infinite number of bits are available, these two functions are equivalent.

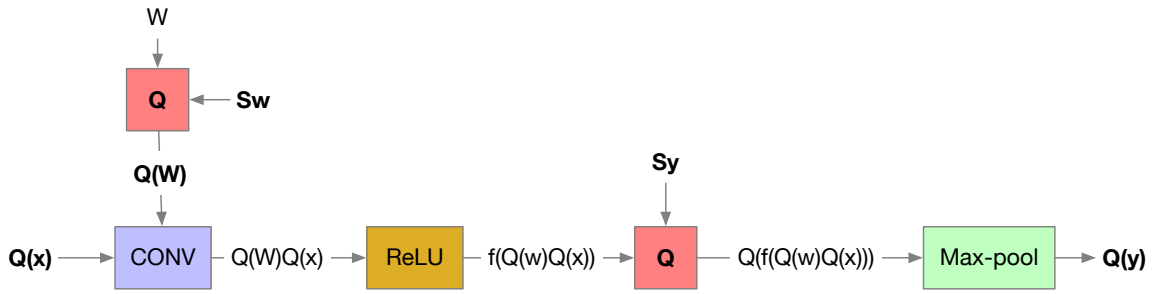
Different input data will result in a different scaling factor S_y^i for the i^{th} layer; however, when deploying the model, each layer needs to have a fixed scaling factor. One could look at all the known input data and use the most conservative scaling factor to avoid any pos-

sible overflow problem. However, this practice almost always results in a too conservative choice that sacrifices a significant amount of resolution for some rarely occurring extreme values. A reasonable amount of clipping is instead allowable, but it is hard to know in advance the exact amount of clipping which can be tolerated. In fact, these scaling factors on each layer can also be learned during the fine-tuning process. Similar to the sample mean and sample standard deviation in the BN layer, we use an exponential moving average to record the instantaneous scaling factor each time data pass through the quantized model. Once the fine-tuning is finished, the exponential moving average has seen the entire training set multiple times with different combinations. Each time, the exponential moving average records the most conservative choice on the current training batch with $\alpha = 0.3$. When a small portion of those training data has large magnitudes, it only affects the current batch. In the long run, the exponential moving average will smooth the overall scaling factor estimation, allowing a few overflows on some batches while being conservative for most batches. This behavior tries to find the optimal scaling factor that uses the dynamic range most effectively without explicitly validating all possible scaling factor choices on the entire training dataset. Some signal saturation due to the overflow effect is acceptable here because the output activation map of the convolutional layer will be passed into another nonlinear ReLU activation function. The existence of these nonlinearities is expected by the CNN model. Thus, the degradation of performance can be partially recovered via updating the weight values during fine-tuning process.

Another challenge for fine-tuning the CPIC model, in particular, is the BN layer. Since the BN layer will be absorbed into the previous CONV layer, the quantization of the combined CONV layer needs to be simulated during the training process. Both μ and σ need to be estimated for each training batch, so the CONV layer needs to be run twice on each iteration. As shown in Figure 3.15a, during the training process, feature maps are passed into the CONV layer with full precision weights followed by the BN layer. This is used to estimate the sample mean μ and sample standard deviation σ as accurately as possible. The



(a) Training



(b) Inference

Figure 3.15: Fine-tuning process for quantized CNN model with the BN layer incorporated into the CONV layer.

estimated μ and σ are then used to simulate the BN layer value for absorption. The new CONV layer with the BN layer absorbed is then quantized following the same procedure shown in Figure 3.13. During inference, the new bias b' is added back after the CONV layer whose weights include the absorbed BN layer coefficients as shown in Figure 3.15b.

Using the training and inference scheme shown in Figure 3.15, the simplified CPIC from Section 3.4.3 is quantized into lower precision. Figure 3.16 shows the evolution of the training and validation processes vs. epochs for the quantized model in 8-bit fixed-point. After an initial drop in validation accuracy, the fine-tuning process gradually recovers most of this loss of classification accuracy due to quantization in about 70 epochs. So, the fine-tuning processing was run for 100 epochs instead 200.

Table 3.7 summarizes results for different numbers of bits: the simplified floating-point model is quantized into dynamic fixed-point numbers using 12, 10, 8, 6, and 4 bits. The most recent high-end seismic sensors record waveforms with 24-bit ADCs, while most inexpensive devices record with at least 12 bits. [TODO] say what we stopped at 12 bit. The

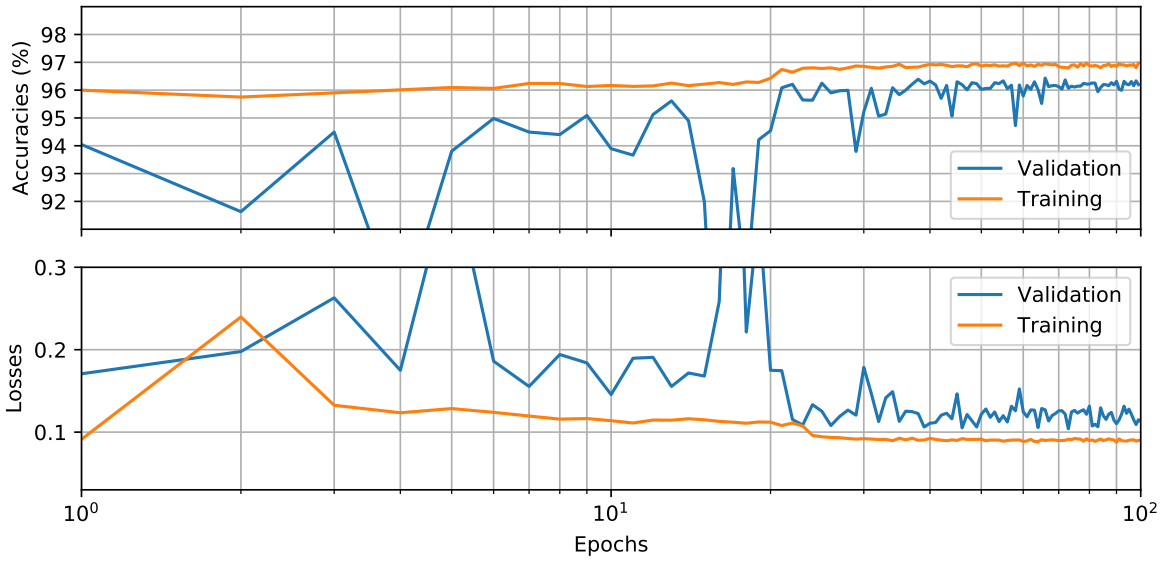


Figure 3.16: Training performance of simplified CPIC model quantized with 8-bit fixed-point numbers on Wenchuan dataset: (a) classifier accuracy and (b) loss function against number of epochs for training and validation datasets during the CNN training process.

first row of Table 3.7 shows the classification accuracy difference between the simplified CPIC model in floating-point and the lower-precision model with only the filter weights quantized. This is a reference point for other quantization experiments. If the assumption that the quantized model is similar to the floating-point case holds, the fine-tuned model should never have classification accuracy higher than the ones shown in the first row. This is true for all but the 4-bit case where the fine-tuned model performs significantly better than the weight-only quantized model. In this case, the weights in the fine-tuned model must be dramatically different from the original floating-point numbers making it inappropriate to use the proposed scheme to fine-tune the model for classification recovery. Indeed, the step-function-shaped 4-bit quantizer is significantly different from a 45-degree line making the approximation shown in Figure 3.14 invalid. Notice that the classification accuracy is better without fine-tuning in the case of 12 bits. This agrees with our assumption that the optimal quantized model is close to the floating-point model given enough number precision.

The fully quantized models are shown in the second row of Table 3.7. The value is

Table 3.7: Validation accuracy of the quantized CPIC fixed-point model with a varying number of bits. Cases with less than 1% accuracy differences are marked in green; those with differences between 1% and 5% in orange; those with differences over 5% in red. The first row is unrealistic where filter weights are quantized but the feature maps are not, but it serves as a reference.

Number of Bits	12	10	8	6	4
Weight-only quantization (%)	97.22	97.16	97.08	95.97	73.54
Fully quantized model (%)	97.18	96.79	95.25	77.93	56.82
Fine-tuned one epoch (%)	96.95	96.76	94.66	89.46	69.89
Fine-tuned 100 epochs (%)	97.13	97.01	96.51	92.70	85.66

slightly inflated as the exponential moving average for scaling factor estimation has been turned off for these models. The scaling factors used in validating them are dynamically computed to avoid any possible overflow problem. Since it is impossible to have a different scaling factor for different input data, the accuracy shown on the second line is slightly better than it should be. After one epoch of training, the exponential moving averages for the scaling factors have seen all training data; so the averages of the scaling factors does not change much for the model on the third row of Table 3.7. Notice that the fine-tuning process completely recovers the classification accuracy for a 12-bit model. The 10-bit and 8-bit cases can be recovered with less than 1% difference in classification accuracy. However, the 6-bit model can only be recovered within a 5% difference making it borderline for feasible usage. The 4-bit model, as discussed before, is not appropriate for the proposed fine-tuning process and has more than a 10% drop in classification accuracy even after the fine-tuning process.

Currently, the quantization process is simulated on GPUs in floating-point with artificially reduced precision. Thus it can correctly model the noise and classification error due to reduced precision; however, the simulated fine-tuning process for the quantized model is much slower than its counterpart for a floating-point model. As more and more general purpose GPUs start to support half-precision, or even quarter-precision integers, this fine-tuning may be conducted with actual fixed-point numbers, which would significantly

improve its speed. Moreover, when the embedded processors start to support the acceleration of matrix-matrix multiplication of low-precision integers, we may even be able to fine-tune the CPIC model directly on the embedded processor of an edge device.

3.6 Deployment on Embedded Devices

The simplified CPIC model quantized in dynamic fixed-point is now ready for deployment on embedded devices. Due to the lack of compatible firmware on the Raspberry Pi board, the evaluation of this deployment of quantized models must be deferred to future work. Note that this thesis provides all the necessary tools for converting a validated floating-point model to the target fixed-point model. In the this section, the floating-point number models are benchmarked instead.

The original CPIC model and simplified CPIC models from Section 3.4 are tested on a four-minute segment of 3-C waveforms from the Mariana dataset. As shown in Figure 3.17, both P and S-wave characteristic functions from the two models have a peak near the corresponding manual pick times. Although it is counterintuitive, arrivals of P-waves are more challenging to pick on the Mariana dataset than those of S-waves. This is primarily due to the source mechanism in the Mariana subduction zone which produces P-wave arrivals with gradually increasing magnitude instead of more abrupt changes from the noise floor. This is very different from cases in the Wenchuan dataset that the CPIC model was originally trained on. Thus, having the CPIC model fine-tuned on a small subset of the Mariana dataset should help with the picking results. Indeed, when fine-tuned on 2,000 samples from the Mariana dataset, the simplified CPIC model shows a significant improvement in picking accuracy for P-waves as shown in Figure 3.17(f). However, this is not true for the original CPIC model. When fine-tuned on 2,000 samples, the picking results hardly change for any of the predicted P-wave arrival times in Figure 3.17(c). We do observe some improvement in denying surface waves that come after S-waves, which were mistakenly recognized as S-waves previously without fine-tuning. After expanding

Table 3.8: Benchmark of CPIC model computation times on Core i7 laptop vs. Raspberry Pi device processing a four-minute three-channel waveform from Mariana dataset.

Model	Original CPIC	Fine-tuned Original CPIC	Simplified CPIC	Fine-tuned Simplified CPIC
i7 Laptop	0.65 s	0.55 s	0.26 s	0.26 s
Raspberry Pi	66.47 s	66.68 s	34.09 s	37.68 s

the number of samples for fine-tuning to 14,000, the picking results in Figure 3.17(d) show some improvement for P-waves. This phenomenon may be caused by the difference in the model sizes. The original CPIC model has more than ten times the number of parameters compared to the simplified model, which means it naturally requires more training samples to reach a stable model during the fine-tuning process.

When comparing the computation time used for processing such four-minute waveform, the simplified model is consistently faster than the original CPIC model. Shown in Table 3.8, the time consumed by simplified model is about half of that by original CPIC model for both i7 laptop and Raspberry Pi device. This makes the simplified model more favorable. Notice that there is no GPU used for both testing cases. Computation time on Raspberry Pi is roughly 100 times of that on the i7 laptop since both the CPU clock frequency and the RAM speed on the Raspberry Pi is considerable smaller than those on the i7 laptop. However, since the computation times on both cases are significantly less than the signal duration, it is feasible to implement a real-time processing system both i7 processors as well as the Raspberry Pi devices.

Although modern hardware accelerators for floating-point arithmetic usually claim comparable efficiency to their fixed-point counterparts, the fixed-point accelerators are still cheaper to implement, consume less power, and are more ubiquitous on low-end embedded devices. Since seismic waveforms are originally recorded in fixed-point numbers, processing the data in fixed-point also has the advantage of avoiding continuously converting input waveforms from fixed-point numbers to floating-point numbers. Finally, notice that many fixed-point processors have the basic processor working at 4-bit word length. Higher precision bit computation is conducted by combining multiple basic processors. This makes

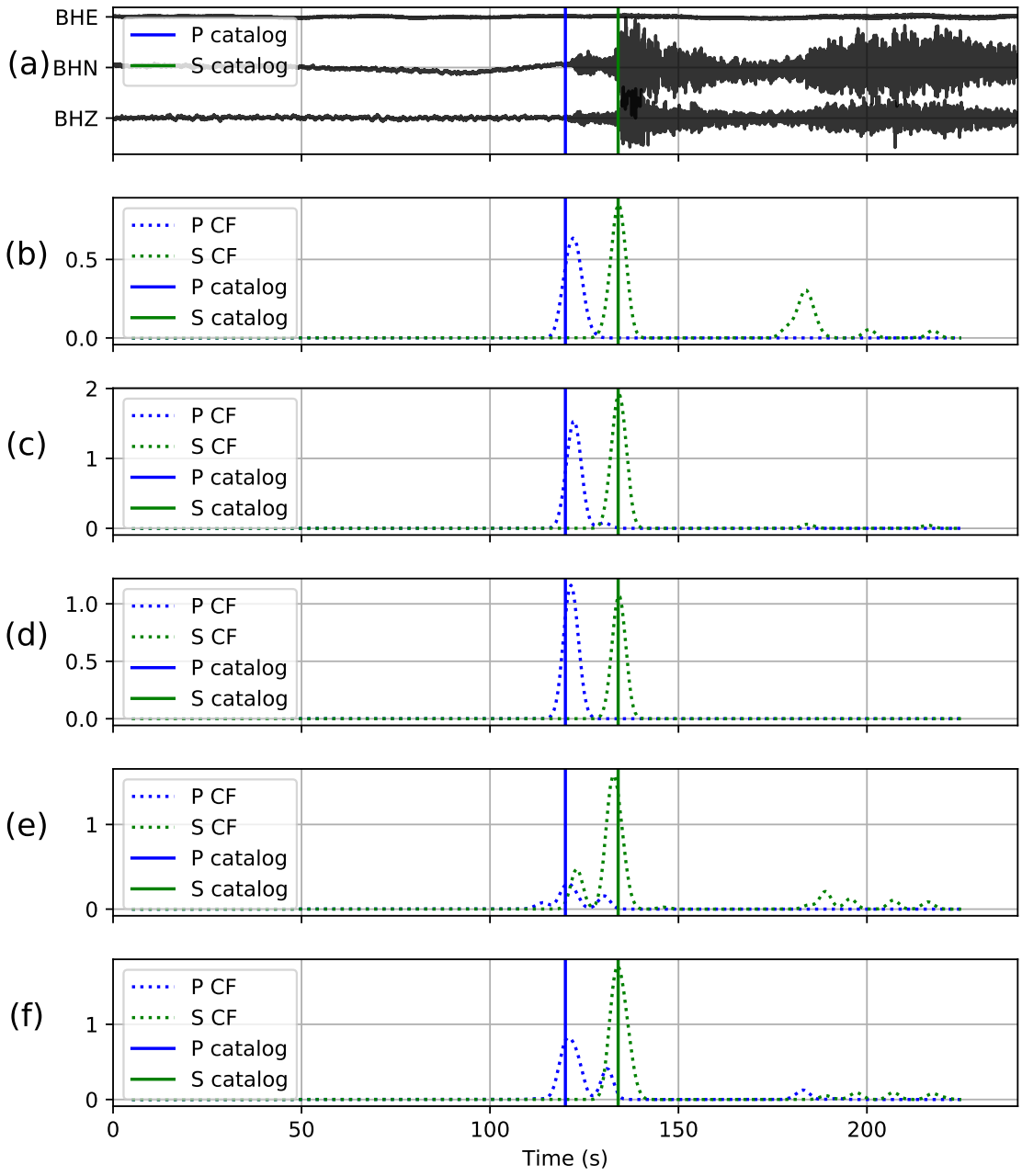


Figure 3.17: Picking results for five CPIC models on a four-minute 3-C waveform from Mariana dataset. The plots from top to bottom panels are (a) three-component waveform; P-wave and S-wave picking characteristic functions of (b) original CPIC model trained on Wenchuan dataset; (c) original CPIC model fine-tuned on 2,000 samples from Mariana dataset; (d) original CPIC model fine-tuned on 16,000 samples from Mariana dataset; (e) simplified CPIC model trained on Wenchuan dataset; and (f) simplified CPIC model fine-tuned on 2,000 samples from Mariana dataset. Note that the vertical blue and green lines indicate manually picked arrival times for P-wave and S-wave, respectively.

selecting the number of quantization bits favor using a multiple of four, e.g., 12-bit and 8-bit choices in Table 3.7 would be preferred.

CHAPTER 4
CONCLUSION AND FUTURE WORK

Neural networks have proven useful in modern data processing and artificial intelligence systems, where successes are rife in various applications, such as computer vision, speech recognition, and robotics. Most existing neural network models are designed for powerful computing servers in the cloud with abundant labeled samples to train on. However, the emergence of IoT and portable, inexpensive, ubiquitous sensors demands that the neural network approach be adapted to smaller training sets with a limited number of labeled samples and to less powerful processors on embedded devices. This thesis is dedicated to bridging the gap between existing large neural network models on GPU servers and the demands of light-weighted small models on the edge devices for the specific application of seismic event detection and phase picking. It lays down the fundamental elements for embedded device deployment of neural networks from two perspectives: training neural networks with limited labeled samples; and simplifying them for devices with limited computing resources and battery power.

Historical milestones in related research are reviewed in Section 1.1 which leads into the neural network system design and deployment further explained in Chapters 2 and 3, respectively. Dating back to the 1940s, the development of neural network approaches is explained in Section 1.1.1 with important techniques introduced. The perceptron concept along with the backpropgation method used to efficiently train MLP model are the foundation of many modern neural network structures. The CNN and training techniques associated with it, e.g. ReLU activation and BN layer, are also described. On the other hand, the general processing pipeline of seismic event detection and phase picking are summarized in Section 1.1.2. Popular event detection and phase picking methods are reviewed, including STA/LTA, matched filter, and most recently neural network approaches. The developments

in both fields urge for reliable approaches to use NN models on a battery-powered embedded devices. One potential design of such system is given in Section 1.2 and its advantages and limitations are explored.

Chapter 2 focuses on one recent CNN model for seismic processing, namely CPIC, as an example to demonstrate the concerns and challenges in designing neural network models with limited labeled samples for training. Various datasets with large, medium, and small sizes are trained and validated using the CPIC model in Section 2.2. Minimum pre-processing, i.e., only soft-clipping, is used for CPIC, whose advantage is explained in Section 2.3. A special CF is proposed as post-processing in Section 2.4 for detecting seismic events and picking phase arrival times on a time sequence of CNN outputs from moving windows. Section 2.5 presents benchmarks with traditional detection and time picking algorithms, as well as other recent neural network approaches, and shows that the proposed CPIC model has great potential for deployment on embedded devices. Generality of the CPIC model is validated on smaller and larger training datasets in Section 2.6. Two examples of transfer learning on an induced seismicity dataset in Oklahoma, and an aftershock dataset in the southern Pacific are also given. Other neural network structures from the recent literature, such as the CNN classifier, the RNN sequence predictor with LSTM, and the CNN autoencoder, are examined and compared for the detection and picking task in Section 2.7. The design techniques and philosophy presented in Chapter 2 are applicable to similar designs with the same neural network components and can be easily extended to other structures with additional components.

In preparation for the embedded device deployment, the CPIC model is further simplified in Chapter 3. By visualizing the filter weights and kernels in the CPIC models in Section 3.3, two simplification approaches are proposed, namely reducing number of parameters and reducing the arithmetic precision during computation. Section 3.4 demonstrates a simplified CPIC model with fewer layers, as well as fewer filters in each layer. The simplified model reduces the number of trainable parameter by almost a factor of 12 with

only a 0.2% drop in classification accuracy. Quantization of the simplified CPIC model is considered in Section 3.5. Using the dynamic fixed-point representation, the classification performance of the CPIC model can be well preserved when using 7-bit integer arithmetic. A fine-tuning process is also proposed which further reduces the necessary number of bits for fixed-point representation down to four bits. The model simplification and fine-tuning techniques are not limited to the CPIC model presented in this thesis, but are generally applicable for a large variety of neural networks with similar computing structures.

Based on this thesis work, the quantized CPIC model can be deployed on embedded processors, such as a Raspbery Pi, with the hardware acceleration provided by a SIMD coprocessor, such as the Arm Neon. On board training of NN models is also possible by using accelerators such as Intel Movidius neural compute stick (NCS), Google Edge TPU, or Nvidia Tegra series chips. The recently released Raspberry Pi 4 has been upgraded to a 1.5 GHz Quad-core ARM Cortex-A72 SoC with 4 GB RAM, which should be sufficient to support small scale fine-tuning tasks. Lower power in-field communication, such as Sub 1 GHz advocated by Texas Instruments, can extend the communication distance between IoT sensors as well as their battery life. Once the hardware advances are all integrated, the overall system presented in Section 1.3 can be completed. Such a system will support accurate real-time event detection and phase picking on a large scale with low cost which is essential for effective EEW systems. The in-field communication can also facilitate distributed computing schemes for arrival time association and event locations. When the arrival times are confirmed via cloud or distributed computing, the CPIC model on each device is updated individually by on-board fine-tuning to increase each node's classification accuracy. The overall system has the potential to significantly reduce the cost of long-term seismic monitoring systems, as well as increase their accuracy and effectiveness.

REFERENCES

[1] R. V. Allen, "Automatic earthquake recognition and timing from single traces," *Bulletin of the Seismological Society of America*, vol. 68, no. 5, pp. 1521–1532, 1978.

[2] Y. Morita and H. Hamaguchi, "Automatic detection of onset time of seismic waves and its confidence interval using the autoregressive model fitting," *Zisin (J. Seism. Soc. Japan)*, vol. 37, pp. 281–293, 1984.

[3] R. Sleeman and T. van Eck, "Robust automatic P-phase picking: An on-line implementation in the analysis of broadband seismogram recordings," *Physics of the earth and planetary interiors*, vol. 113, pp. 265–275, Jun. 1999.

[4] Z. E. Ross, M.-A. Meier, E. Hauksson, and T. H. Heaton, "Generalized seismic phase detection with deep learning," *Bulletin of the Seismological Society of America*, vol. 108, no. 5A, pp. 2894–2901, 2018.

[5] L Zhu, Z Peng, and J McClellan, "Event detection and phase picking based on deep convolutional neural networks," in *80th EAGE Conference and Exhibition 2018*, 2018.

[6] W. Zhu and G. C. Beroza, "Phasenet: A deep-neural-network-based seismic arrival-time picking method," *Geophysical Journal International*, vol. 216, no. 1, pp. 261–273, 2019.

[7] L. Zhu, Z. Peng, and J. McClellan, "Deep learning for seismic event detection of earthquake aftershocks," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, IEEE, 2018, pp. 1121–1125.

[8] L. Zhu, Z. Peng, J. McClellan, C. Li, D. Yao, Z. Li, and L. Fang, "Deep learning for seismic phase detection and picking in the aftershock zone of 2008 Mw7.9 wenchuan earthauqke," *Physics of the Earth and Planetary Interiors*, 2019.

[9] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[10] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Oxford, England: Wiley, 1949.

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[12] B Widrow, *An adaptive'adaline'neuron using chemical'memistors', 1553-1552*, 1960.

[13] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[15] B. Widrow and M. E. Hoff, "Associative storage and retrieval of digital information in networks of adaptive "neurons"," in *Biological Prototypes and Synthetic Systems*, Springer, 1962, pp. 160–160.

[16] C. R. Winter and B Widrow, "Madaline rule ii: A training algorithm for neural networks," in *Second Annual International Conference on Neural Networks*, 1988, pp. 1–401.

[17] D. Andes, "Mriii: A robust algorithm for training analog neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 1990.

[18] N. Rochester, J Holland, L Haibt, and W Duda, "Tests on a cell assembly theory of the action of the brain, using a large digital computer," *IRE Transactions on information Theory*, vol. 2, no. 3, pp. 80–93, 1956.

[19] M. Minsky, "A neural-analogue calculator based upon a probability model of reinforcement," *Harvard University Psychological Laboratories, Cambridge, Massachusetts*, 1952.

[20] J. Von Neumann, *Non-linear capacitance or inductance switching, amplifying, and memory organs*, 1957.

[21] ——, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[22] M. Minsky and S. Papert, "Perceptron: An introduction to computational geometry," *The MIT Press, Cambridge, expanded edition*, vol. 19, no. 88, p. 2, 1969.

[23] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences//phd thesis, dept. of applied mathematics. harvard university, cambridge, ma., 1974," 1974.

[24] D Parker, "Learning logic. invention report 581-64, department of electrical engineering," *Stanford University, Stanford*, vol. 581, p. 64, 1982.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[27] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[28] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *Backpropagation: Theory, Architectures and Applications*, pp. 35–61, 1995.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4-5, pp. 291–294, 1988.

[31] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[32] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in neural information processing systems*, 1994, pp. 3–10.

[33] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[34] G. A. Carpenter, S. Grossberg, *et al.*, "The art of adaptive pattern recognition by a self-organizing neural network," *IEEE computer*, vol. 21, no. 3, pp. 77–88, 1988.

[35] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[36] R. M. Neal, "Connectionist learning of belief networks," *Artificial intelligence*, vol. 56, no. 1, pp. 71–113, 1992.

[37] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The helmholtz machine," *Neural computation*, vol. 7, no. 5, pp. 889–904, 1995.

[38] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The "wake-sleep" algorithm for unsupervised neural networks," *Science*, vol. 268, no. 5214, pp. 1158–1161, 1995.

[39] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[40] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[41] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[42] S. Thrun, "Learning to play the game of chess," in *Advances in neural information processing systems*, 1995, pp. 1069–1076.

[43] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Temporal difference learning of position evaluation in the game of go," in *Advances in Neural Information Processing Systems*, 1994, pp. 817–824.

[44] G. Hinton, *Recent developments in deep learning*, Youtube, 2013.

[45] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[46] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[47] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[48] Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[49] A.-r. Mohamed, T. N. Sainath, G. E. Dahl, B. Ramabhadran, G. E. Hinton, M. A. Picheny, *et al.*, "Deep belief networks using discriminative features for phone recognition.," in *ICASSP*, 2011, pp. 5060–5063.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[51] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 873–880.

[52] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[53] K. Jarrett, K. Kavukcuoglu, Y. LeCun, *et al.*, "What is the best multi-stage architecture for object recognition?" In *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 2146–2153.

[54] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[55] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[56] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[57] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, 2015, pp. 448–456.

[59] Q. Kong, D. T. Trugman, Z. E. Ross, M. J. Bianco, B. J. Meade, and P. Gerstoft, "Machine learning in seismology: Turning data into insights," *Seismological Research Letters*, vol. 90, no. 1, pp. 3–14, 2018.

[60] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[61] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[62] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[64] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, vol. 4, 2017, p. 12.

[65] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[66] R. Allen, "Automatic phase pickers: Their present use and future prospects," *Bulletin of the Seismological Society of America*, vol. 72, no. 6B, S225–S242, Dec. 1982.

[67] M Baer and U Kradolfer, "An automatic phase picker for local and teleseismic events," *Bulletin of the Seismological Society of America*, vol. 77, no. 4, pp. 1437–1445, 1987.

[68] H. Akaike, "Markovian representation of stochastic processes and its application to the analysis of autoregressive moving average processes," *Annals of the Institute of Statistical Mathematics*, vol. 26, no. 1, pp. 363–387, 1974.

[69] V. Pisarenko, A. Kushnir, and I. Savin, "Statistical adaptive algorithms for estimation of onset moments of seismic phases," *Physics of the Earth and Planetary Interiors*, vol. 47, pp. 4 –10, 1987.

[70] T. Takanami and G. Kitagawa, "Estimation of the arrival times of seismic waves by multivariate time series model," *Annals of the Institute of Statistical Mathematics*, vol. 43, no. 3, pp. 407–433, 1991.

[71] M. Tarvainen, "Automatic seismogram analysis: Statistical phase picking and locating methods using one-station three-component data," *Bulletin of the Seismological Society of America*, vol. 82, no. 2, p. 860, 1992.

[72] C. D. Saragiotis, L. J. Hadjileontiadis, and S. M. Panas, "PAI-S/K: a robust automatic seismic P phase arrival identification scheme," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 6, pp. 1395–1404, 2002.

[73] S. E. J. Nippress, A. Rietbrock, and A. E. Heath, "Optimized automatic pickers: Application to the ANCORP data set," *Geophysical Journal International*, vol. 181, no. 2, pp. 911–925, 2010.

[74] A. Jurkevics, "Polarization analysis of three-component array data," *Bulletin of the Seismological Society of America*, vol. 78, no. 5, pp. 1725–1743, 1988.

[75] A. Cichowicz, "An automatic S-phase picker," *Bulletin of the Seismological Society of America*, vol. 83, no. 1, pp. 180–189, 1993.

[76] A. Rosenberger, "Real-time ground-motion analysis: Distinguishing P and S arrivals in a noisy environment," *Bulletin of the Seismological Society of America*, vol. 100, no. 3, pp. 1252–1262, 2010.

[77] I Kurzon, F. Vernon, A Rosenberger, and Y Ben-Zion, "Real-time automatic detectors of P and S waves using singular value decomposition," *Bulletin of the Seismological Society of America*, vol. 104, no. 4, pp. 1696–1708, 2014.

[78] Z. Li, Z. Peng, D. Hollis, L. Zhu, and J. McClellan, "High-resolution seismic event detection using local similarity for large-n arrays," *Scientific reports*, vol. 8, no. 1, p. 1646, 2018.

[79] S. Rost and C. Thomas, "Array seismology: Methods and applications," *Reviews of Geophysics*, vol. 40, no. 3, pp. 2–1–2–27, 2002.

[80] J. Almendros, J. M. Ibáñez, G. Alguacil, and E. Del Pezzo, "Array analysis using circular-wave-front geometry:an application to locate the nearby seismo-volcanic source," *Geophysical Journal International*, vol. 136, no. 1, pp. 159–170, 1999.

[81] S. J. Gibbons and F. Ringdal, "The detection of low magnitude seismic events using array-based waveform correlation," *Geophysical Journal International*, vol. 165, no. 1, pp. 149–166, 2006.

[82] D. R. Shelly, G. C. Beroza, and S. Ide, "Non-volcanic tremor and low-frequency earthquake swarms," *Nature*, vol. 446, 305–307, 2007.

[83] Z. Peng and P. Zhao, "Migration of early aftershocks following the 2004 Parkfield earthquake," *Nature Geoscience*, vol. 2, no. 12, pp. 877–881, 2009.

[84] J. R. Brown, G. C. Beroza, and D. R. Shelly, "An autocorrelation method to detect low frequency earthquakes within tremor," *Geophysical Research Letters*, vol. 35, no. 16, 2008.

[85] A. C. Aguiar and G. C. Beroza, "Pagerank for earthquakes," *Seismological Research Letters*, vol. 85, no. 2, pp. 344–350, 2014.

[86] J. Zhang, H. Zhang, E. Chen, Y. Zheng, W. Kuang, and X. Zhang, "Real-time earthquake monitoring using a search engine method," *Nature communications*, vol. 5, p. 5664, 2014.

[87] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford InfoLab, Tech. Rep., 1999.

[88] S. Baluja and M. Covell, "Content fingerprinting using wavelets," in *The 3rd European Conference on Visual Media Production (CVMP 2006) - Part of the 2nd Multimedia Conference 2006*, 2006, pp. 198–207.

[89] C. E. Yoon, O. O'Reilly, K. J. Bergen, and G. C. Beroza, "Earthquake detection through computationally efficient similarity search," *Science Advances*, vol. 1, no. 11, 2015.

[90] L. Zhu, Z. Li, Z. Peng, E. Liu, and J. H. McClellan, "Weighted random sampling in seismic event detection/location (WRASED): Applications to local, regional, and global seismic networks," *Seismological Research Letters*, vol. 88, no. 2B, pp. 463–723, 2017.

[91] T. Perol, M. Gharbi, and M. Denolle, "Convolutional neural network for earthquake detection and location," *Science Advances*, vol. 4, no. 2, 2018.

[92] M. D. McCormack, D. E. Zaucha, and D. W. Dushek, "First-break refraction event picking and seismic data trace editing using neural networks," *Geophysics*, vol. 58, no. 1, pp. 67–78, 1993.

[93] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[94] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon, and A. RABKin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[95] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[96] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[97] F. Sepulveda and J. Pulliam, "The internet of geophysical things: Raspberry pi enhanced REF TEK (RaPiER) system integration and evaluation," *Seismological Research Letters*, vol. 87, no. 2A, pp. 345–357, 2016.

[98] C. Li, L. Zhu, D. Yao, X. Meng, Z. Peng, J. H. McClellan, and J. I. Walter, "Transfer learning for seismic phase picking on different geographic regions," in *AGU Fall Meeting Abstracts*, 2018.

[99] L. Zhu, C. Li, Z. Li, Z. Peng, and J. McClellan, "Lightweight cnn-based real-time phase picker embedded in seismic sensors," in *AGU Fall Meeting Abstracts*, 2018.

[100] Y. Zhou, H. Yue, Q. Kong, and S. Zhou, "Hybrid event detection and phase-picking algorithm using convolutional and recurrent neural networks," *Seismological Research Letters*, 2019.

[101] Y. T. Zhou and R. Chellappa, "Computation of optical flow using a neural network," in *IEEE 1988 International Conference on Neural Networks*, 1988, 71–78 vol.2.

[102] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[103] L. Fang, Z. Wu, and K. Song, "SeismOlympics," *Seismological Research Letters*, vol. 88, no. 6, pp. 1429–1430, 2017.

[104] Y. Y. Kagan, "Short-term properties of earthquake catalogs and models of earthquake source," *Bulletin of the Seismological Society of America*, vol. 94, no. 4, pp. 1207–1228, 2004.

[105] Z. Peng, J. E. Vidale, and H. Houston, "Anomalous early aftershock decay rate of the 2004 Mw 6.0 Parkfield, California, earthquake," *Geophysical Research Letters*, vol. 33, no. 17, 2006.

[106] F. Waldhauser and W. L. Ellsworth, "A double-difference earthquake location algorithm: Method and application to the northern hayward fault, california," *Bulletin of the Seismological Society of America*, vol. 90, no. 6, pp. 1353–1368, 2000.

[107] X. Xu, X. Wen, G. Yu, G. Chen, Y. Klinger, J. Hubbard, and J. Shaw, "Coseismic reverse- and oblique-slip surface faulting generated by the 2008 Mw 7.9 Wenchuan earthquake, China," *Geology*, vol. 37, no. 6, pp. 515–518, 2009.

[108] G. Feng, E. A. Hetland, X. Ding, Z. Li, and L. Zhang, "Coseismic fault slip of the 2008 Mw 7.9 Wenchuan earthquake estimated from InSAR and GPS measurements," *Geophysical Research Letters*, vol. 37, no. 1, 2010.

[109] S. Hartzell, C. Mendoza, L. Ramirez-Guzman, Y. Zeng, and W. Mooney, "Rupture history of the 2008 Mw 7.9 Wenchuan, China, earthquake: Evaluation of separate and joint inversions of geodetic, teleseismic, and strong-motion data," *Bulletin of the Seismological Society of America*, vol. 103, no. 1, pp. 353–370, 2013.

[110] X. Yin, J.-h. Chen, Z. Peng, X. Meng, Q. Liu, B. Guo, and S. Cheng Li, "Evolution and distribution of the early aftershocks following the 2008 Mw 7.9 Wenchuan earthquake in Sichuan, China," *Journal of Geophysical Research: Solid Earth*, vol. 123, no. 9, pp. 7775–7790, Aug. 2018.

[111] G. Zhu, H. Yang, J. Lin, Z. Zhou, M. Xu, J. Sun, and K. Wan, "Along-strike variation in slab geometry at the southern mariana subduction zone revealed by seismicity through ocean bottom seismic experiments," *Geophysical Journal International*, 2019.

[112] X. Chen, J. Haffener, T. H. W. Goebel, X. Meng, Z. Peng, and J. C. Chang, "Temporal correlation between seismic moment and injection volume for an induced earthquake sequence in central oklahoma," *Journal of Geophysical Research: Solid Earth*, vol. 123, no. 4, pp. 3047–3064, 2018.

[113] P. Bird, "An updated digital model of plate boundaries," *Geochemistry, Geophysics, Geosystems*, vol. 4, no. 3, 2003.

[114] B Apolloni *et al.*, "Support vector machines and mlp for automatic classification of seismic signals at stromboli volcano," in *Neural Nets WIRN09: Proceedings of the 19th Italian Workshop on Neural Nets, Vietri Sul Mare, Salerno, Italy May 28-30 2009*, IOS Press, vol. 204, 2009, p. 116.

[115] J. Woollam, A. Rietbrock, A. Bueno, and S. De Angelis, "Convolutional neural network for seismic phase classification, performance demonstration over a local seismic network," *Seismological Research Letters*, vol. 90, no. 2A, pp. 491–502, 2019.

[116] A. Helman, *The Finest Peaks-Prominence and Other Mountain Measures*. Trafford Publishing, 2005.

[117] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Advances in Neural Information Processing Systems 30, Autodiff Workshop*, 2017.

[118] M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, "ObsPy: a python toolbox for seismology," *Seismological Research Letters*, vol. 81, no. 3, pp. 530–533, 2010.

[119] X. Meng, Z. Peng, and J. L. Hardebeck, "Seismicity around Parkfield correlates with static shear stress changes following the 2003 Mw 6.5 San Simeon earthquake," *Journal of Geophysical Research: Solid Earth*, vol. 118, no. 7, pp. 3576–3591, 2013.

[120] S. M. Mousavi, W. Zhu, Y. Sheng, and G. C. Beroza, "Cred: A deep residual network of convolutional and recurrent units for earthquake signal detection," *arXiv preprint arXiv:1810.01965*, 2018.

[121] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.

[122] M. Etienne, C. Ray, and J. N. Thompson, *Simultaneous shooting nodal acquisition seismic survey methods*, 2016.

[123] A. Inbal, R. W. Clayton, and J.-P. Ampuero, "Imaging widespread seismicity at midlower crustal depths beneath long beach, ca, with a dense seismic array: Evidence for a depth-dependent earthquake size distribution," *Geophysical Research Letters*, vol. 42, no. 15, pp. 6314–6323, 2015.

[124] N. Riahi and P. Gerstoft, "The seismic traffic footprint: Tracking trains, aircraft, and cars seismically," *Geophysical Research Letters*, vol. 42, no. 8, pp. 2674–2681, 2015.

[125] D. Slater and D. Hollis, "California's dense urban environment spawns friendlier 3D seismic survey design," *Oil and Gas Journal*, vol. 110, no. 11, p. 54, 2012.

[126] S. M. Hansen and B. Schmandt, "Automated detection and location of microseismicity at mount st. helens with a large-n geophone array," *Geophysical Research Letters*, vol. 42, no. 18, pp. 7390–7397, 2015.

[127] J. R. Sweet, K. R. Anderson, S. Bilek, M. Brudzinski, X. Chen, H. DeShon, C. Hayward, M. Karplus, K. Keranen, C. Langston, *et al.*, "A community experiment to record the full seismic wavefield in oklahoma," *Seismological Research Letters*, vol. 89, no. 5, pp. 1923–1930, 2018.

[128] S. L. Dougherty, E. Cochran, and R. Harrington, "Large-n array observations of injection-induced seismicity in northern oklahoma: The lasso experiment," in *AGU Fall Meeting Abstracts*, 2017.

[129] Q. Kong, R. M. Allen, L. Schreier, and Y.-W. Kwon, "Myshake: A smartphone seismic network for earthquake early warning and beyond," *Science advances*, vol. 2, no. 2, e1501055, 2016.

[130] N. Kurata, "Development of sensor module for seismic and structural monitoring with a chip-scale atomic clock," in *Proceedings of the 16th world conference on earthquake engineering (16WCEE), Paper*, 2017.

[131] E. Upton and G. Halfacree, *Raspberry Pi user guide*. John Wiley & Sons, 2014.

[132] B. Christensen and J. Blanco Chia, "Raspberry shake-a world-wide citizen seismograph network," in *AGU Fall Meeting Abstracts*, 2017.

[133] R. E. Anthony, A. T. Ringler, D. C. Wilson, and E. Wolin, "Do low-cost seismographs perform well enough for your network? an overview of laboratory tests and field observations of the osop raspberry shake 4d," *Seismological Research Letters*, vol. 90, no. 1, pp. 219–228, 2018.

[134] A. Manconi and V. Coviello, "Evaluation of the raspberry shakes seismometers to monitor rock fall activity in alpine environments," in *EGU General Assembly Conference Abstracts*, vol. 20, 2018, p. 16 183.

[135] J. Pulli, "Seismic observations in the washington, DC area with a raspberry shake network," in *AGU Fall Meeting Abstracts*, 2018.

[136] E. Calais, D. Boisson, S. Symithe, R. Momplaisir, C. Prépetit, S. Ulysse, G. P. Etienne, F. Courboulex, A. Deschamps, T. Monfret, *et al.*, "Can a raspberry shake seismic network complement a national seismic network? a case study in haiti," 2019.

[137] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[138] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, Austin, TX, vol. 4, 2010.

[139] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: New features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.

[140] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv: 1410.0759*, 2014.

[141] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[142] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," Tech. Rep., 2011.

[143] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[144] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[145] X. Li, G. Zhang, H. H. Huang, Z. Wang, and W. Zheng, "Performance analysis of gpu-based convolutional neural networks," in *2016 45th International Conference on Parallel Processing (ICPP)*, IEEE, 2016, pp. 67–76.

[146] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[147] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[148] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[149] M. Gevers and G. Li, "Finite word length errors and computations," in *Parametrizations in Control, Estimation and Filtering Problems: Accuracy Aspects*. London: Springer London, 1993, pp. 23–44, ISBN: 978-1-4471-2039-1.

[150] P. Gysel, "Ristretto: Hardware-oriented approximation of convolutional neural networks," *arXiv preprint arXiv:1605.06402*, 2016.

[151] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *arXiv preprint arXiv:1604.03168*, 2016.

[152] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

VITA

Lijun Zhu was born in Suzhou, China in April 1990. He received his B.S. and M.S. in Electrical Engineering, both from Georgia Institute of Technology in 2013 and 2018, respectively. He is expected to receive a Ph.D. degree in August 2019. He is a currently a Ph.D. student at the Center of Energy and Geo Processing (CeGP) in Georgia Institute of Technology under the supervision of Prof. James McClellan. Previously employed by Bose Corporation in 2012, he has extensive knowledge on the audio engineering and acoustic measurements. It was further explored during his employment in Microsoft in 2014, where he developed numerical simulation software for nonlinear ultrasonic acoustic wave propagation. Due to his Ph.D. research, he interned at Aramco Service Company in 2015 for applying machine learning techniques in processing seismic waveform. He brought his knowledge in machine learning to the internship in Texas Instrument in 2018 for developing quantization system of deep neural networks. Lijun's research interests include waveform acquisition technology, numerical simulation algorithm for wave propagation, array-based waveform processing, and deep learning for remote sensing systems.